HZ BOOKS 华章IT

RAW 丛书

站在技术的前沿，紧跟视觉技术发展趋势

计算机视觉增强现实应用程序开发

深圳中科呼图信息技术有限公司 ◎编著

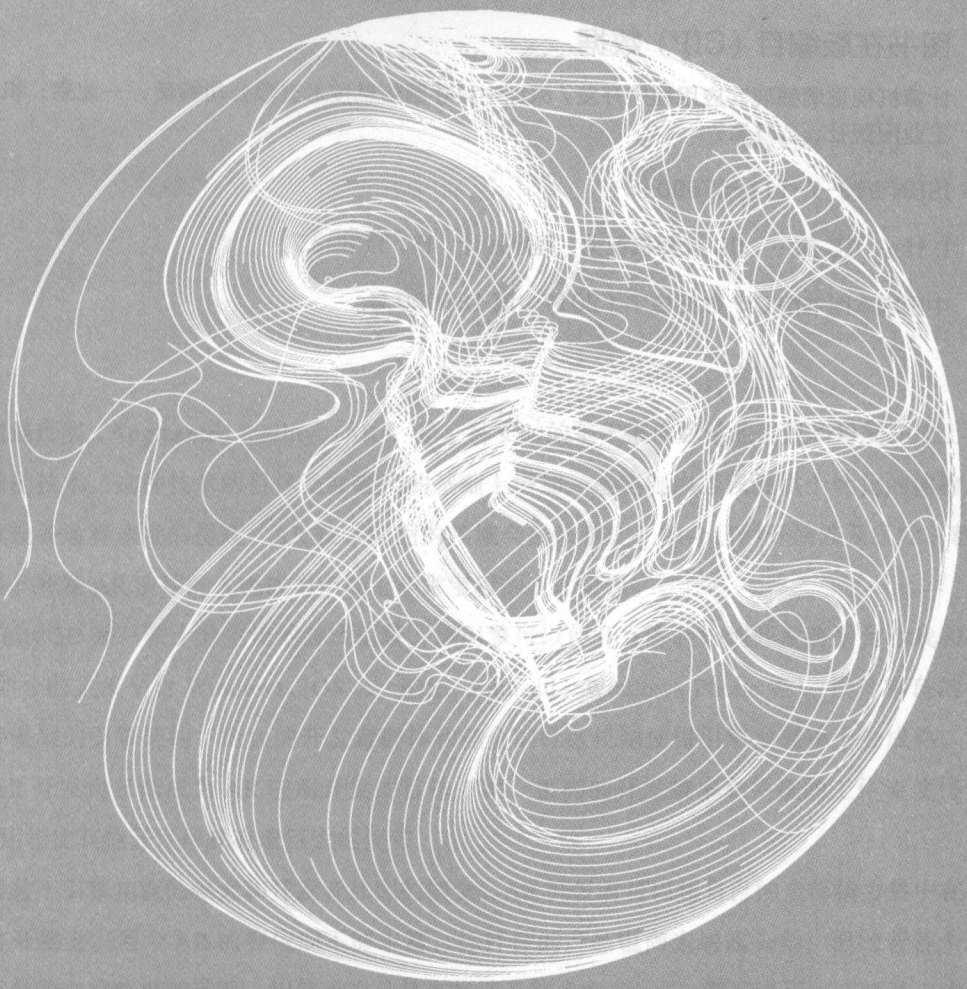
机械工业出版社
China Machine Press

RAVV 简介

RAVV是位于美国硅谷的一家科技产业平台公司，专注于时代前沿的机器人、人工智能、无人驾驶、增强现实、计算机视觉应用等科技，致力于搭建一个以教育、科技、媒体、产品四个基础模块为架构的崭新平台。

RAVV教育的专家组成员囊括了加州大学伯克利分校、加州大学圣克鲁兹分校、香港理工大学、浙江大学等知名学府有关人工智能、计算机视觉应用等科技领域的学术精英。

本书为RAVV教育的教材之一，配套的RAVV课程可以让有志加入AR/VR行业的从业人员更好地增加知识、掌握技术，提高竞争力。



计算机视觉增强现实应用程序开发

深圳中科呼图信息技术有限公司 ◎编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机视觉增强现实应用程序开发 / 深圳中科呼图信息技术有限公司编著 . —北京：机械工业出版社，2017.7

ISBN 978-7-111-57690-7

I. 计… II. 深… III. 计算机视觉 - 程序设计 IV. TP302.7

中国版本图书馆 CIP 数据核字（2017）第 186000 号

计算机视觉增强现实应用程序开发

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：陈佳媛　　责任校对：李秋荣

印　　刷：北京文昌阁彩色印刷有限责任公司　　版　　次：2017 年 9 月第 1 版第 1 次印刷

开　　本：185mm × 260mm　1/16　　印　　张：15.5

书　　号：ISBN 978-7-111-57690-7　　定　　价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88379426　88361066　　投稿热线：（010）88379604

购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 前　言

增强现实（Augmented Reality，AR），是一种实时的基于摄像影像的位置和角度并加上自定义图像的技术，这种技术的目标是在现实的基础上增加一些定制内容以提供个性化的体验。随着计算设备运算速度的提高，AR 技术应用的领域会越来越广，如今已经开始涉及医疗、教育、工业等。开发 AR 应用的人也成为市场上非常宝贵的人才资源。

Unity 3D 是一款十分主流的游戏引擎，该引擎可以使开发者轻松地开发跨平台游戏和应用（支持如今的主流平台 Windows、OS X、iOS、Android、Windows Phone 等），设计精良的松耦合组件模型的引擎架构、庞大的开发者群体和健全的社区机制使得引擎发展得十分成熟。该引擎和 AR 开发也日渐成为 AR 应用开发的首选方案，极高的开发效率和系统的工作流使得开发 AR 应用相比其他开发形式有着巨大的优势。

得益于引擎功能的强大和开发者社区的健全，AR 开发者可以将自己的全部精力集中在项目的内容开发上。当今 AR 内容依旧不够丰富，标杆性的产品也尚未崭露头角，但随着各个厂商对该部分的内容越来越重视，相信无论是从市场还是技术方面，AR 都迎来了一个临界点式的突破，对所有的 AR 开发者来说，这无疑是最好的机会。

本书的目标读者主要分为以下两类。

❑ 初学者

初学者可以通过本书学习基本的编程思路和方法，书中没有长篇大论的理论知识，更多的是从实践出发，在实际应用中了解程序的运行机制、Unity 3D 的结构以及使用程序的思维解决实际问题的方法和经验。完成每章的项目部分后，重点在于理解解决方案的思路。每章都会抛砖引玉地介绍一些计算机的其他领域，有兴趣的读者可以去保存更多感兴趣的领域，来丰富自己的理论知识。当实践能力变得熟练时，理论知识才是限制个人能力最大的障碍。希望初学者可以通过本书先培养对编程领域的兴趣，然后再进一步提高自己感兴趣的具体方向的能力。

❑ 有经验的程序开发者

有经验的开发者可以将本书作为一本熟悉 SDK 操作的参考书阅读。本书包含 Unity 和相关 SDK 的原理介绍，可以让有经验的开发者在最短的时间内了解开发相关内容的方法，提高工作效率和产出。

本书的内容主要包括 Unity 3D 编程基础、Unity 核心组件介绍、常用插件和第三方 SDK 的使用简介及相关的演示项目，项目都有详细的代码和功能介绍，循序渐进地讲解需要使用的工具的基本原理和使用方法，可以让开发者在最短的时间内了解相关领域的开发技巧。希望通过这些项目可以为想要学习 AR 开发的读者提供学习的方向，帮助他们了解实际应用开发时需要考虑的问题以及一些实用的技巧。书中还会涉及一些网络的开发，有兴趣的读者可以以示例项目作为引导更深入和具体地学习网络开发的知识和技巧。

Contents 目　录

第 1 章 Chapter 1

Unity 3D AR 应用开发基础

1.1 Unity 3D 引擎简介

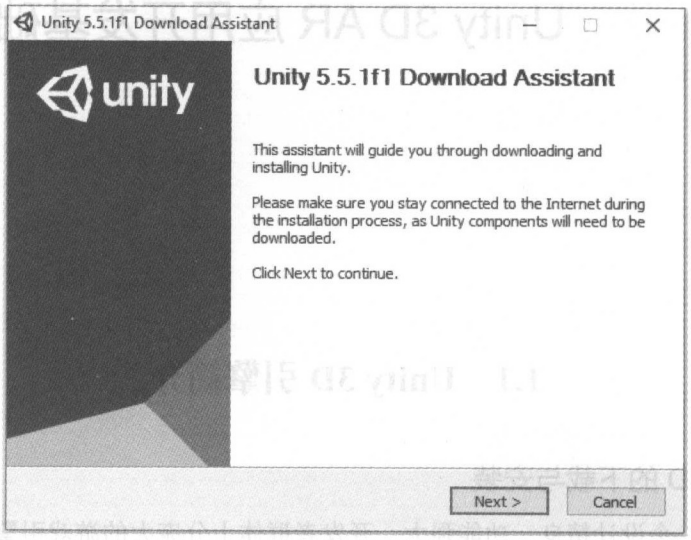
1.1.1 Unity 3D 的下载与安装

Unity3D 是一个设计精良、功能强大、开发者群体十分庞大的游戏引擎。目前官方提供下载安装助手进行下载安装。下载地址为 https://store.unity.com/cn/download。

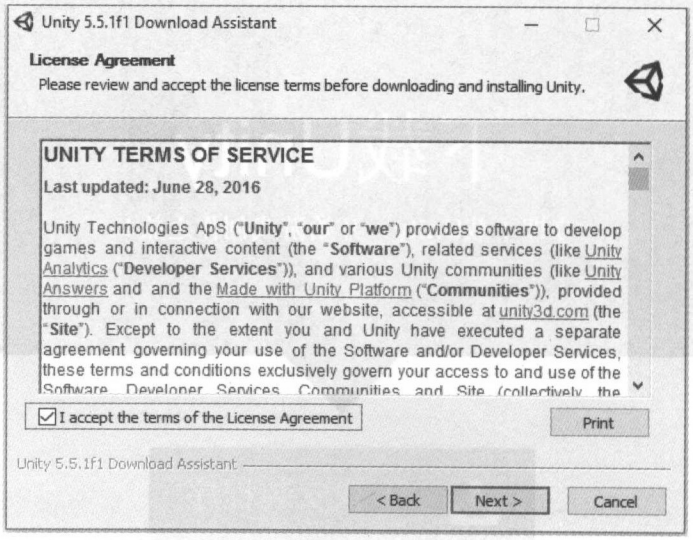
注意官方给出的软硬件平台要求，不满足软硬件条件将无法正常运行 Unity。本书采用 Windows 开发，图形 API 为 DirectX 11。

安装步骤如下：

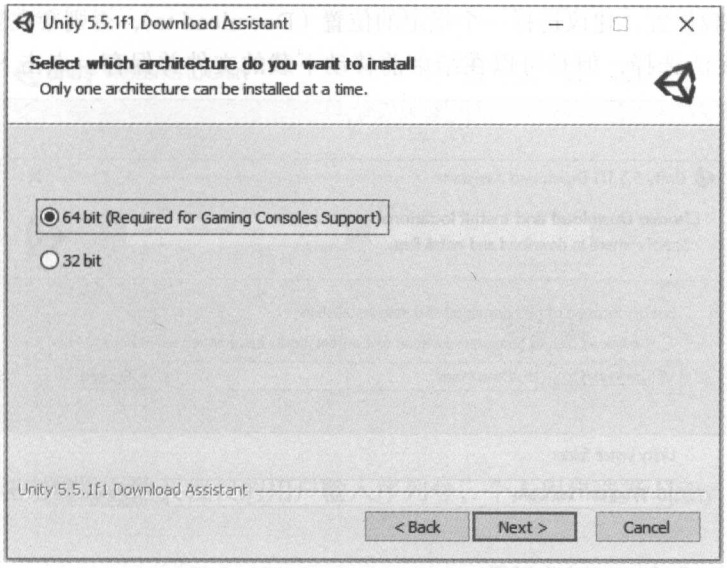
1）下载完成后打开 UnityDownloadAssistant-5.5.1f1.exe（这里的 5.5.1 分别代表大、中、小版本号，f1 代表该版本的第几次发布），点击 Next。



2）选中“I accept the terms of the License Agreement”（我同意授权协议条款），点击 Next。

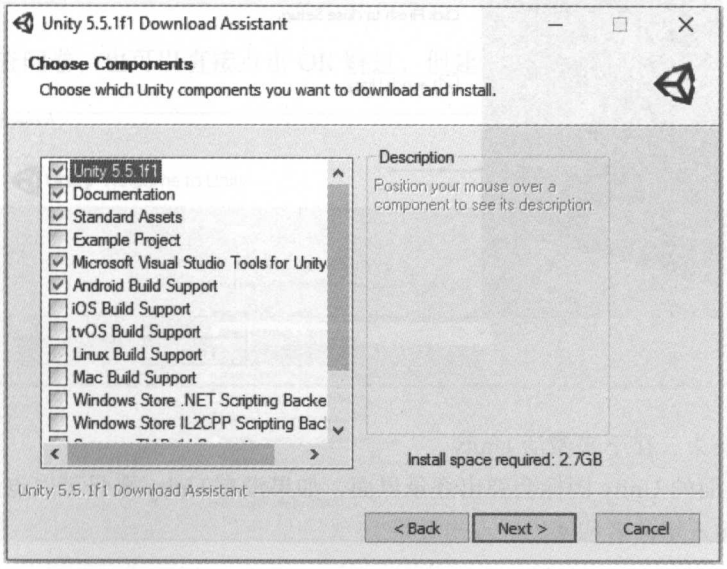


3）建议选择 64 位的环境（可以使用更大的内存并充分利用 64 位处理器的性能，但如果你的操作系统是 32 位的，则只能选择 32 位），点击 Next。

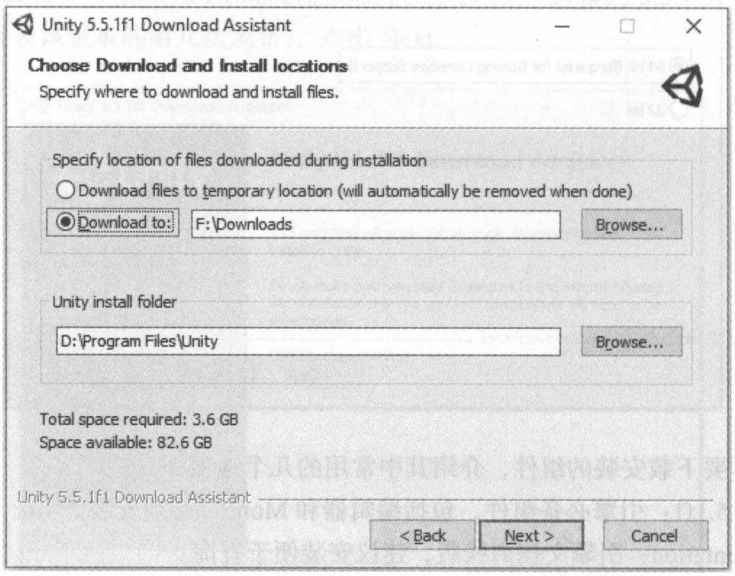


4）选择需要下载安装的组件，介绍其中常用的几个选项：

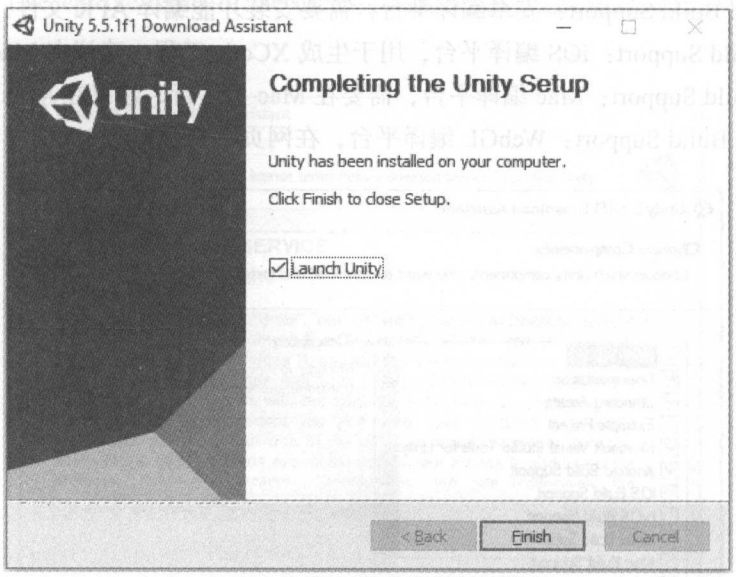
- Unity 5.5.1f1：引擎必备组件，包括编辑器和 Mono，必须安装。
- Documentation：引擎文档离线版，建议安装便于查询。
- Standard Assets：引擎自带标准资源，建议安装方便学习与使用。
- Example Project：工程示例，可以作为参考深入学习 Unity，选择安装。
- Android Build Support：安卓编译平台，需要安装才能编译 APK 文件。
- iOS Build Support：iOS 编译平台，用于生成 XCode 工程，建议在 Mac 下安装。
- Mac Build Support：Mac 编译平台，需要在 Mac 上运行，建议在 Mac 下安装。
- WebGL Build Support：WebGL 编译平台，在网页平台上发布 3D 应用，选择安装。



5）选择下载位置，建议选择一个指定的位置（Download to），否则会在安装结束时删除文件。Mac 版无法选择，但是可以在结束前移动下载的文件并保存。点击 Next，等待下载和安装的过程。

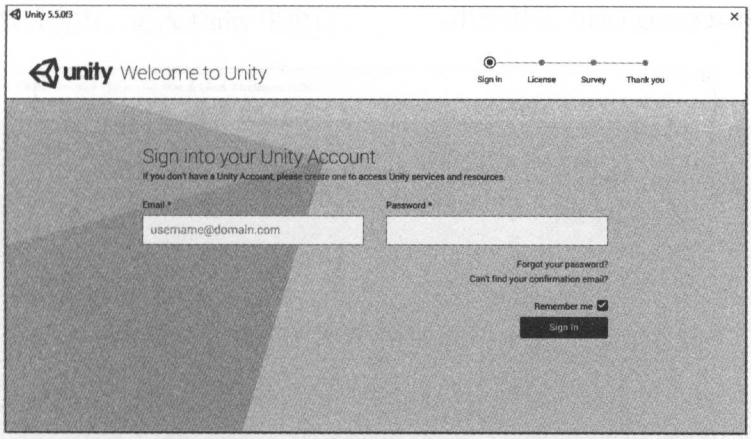


6）安装结束后，点击 Finish 完成安装。

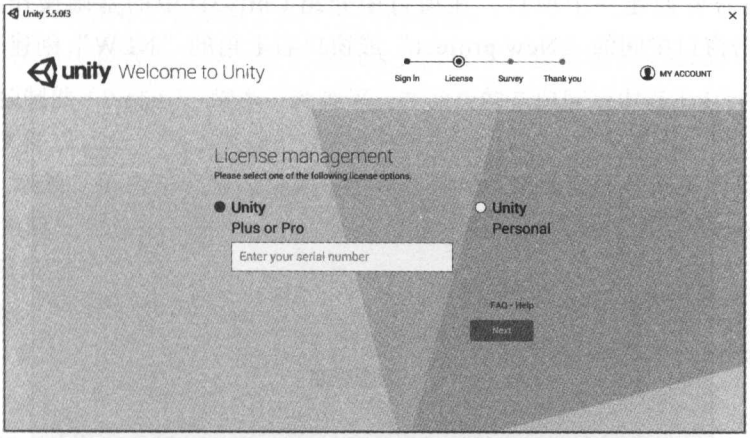


安装到此结束，接下来激活 Unity。

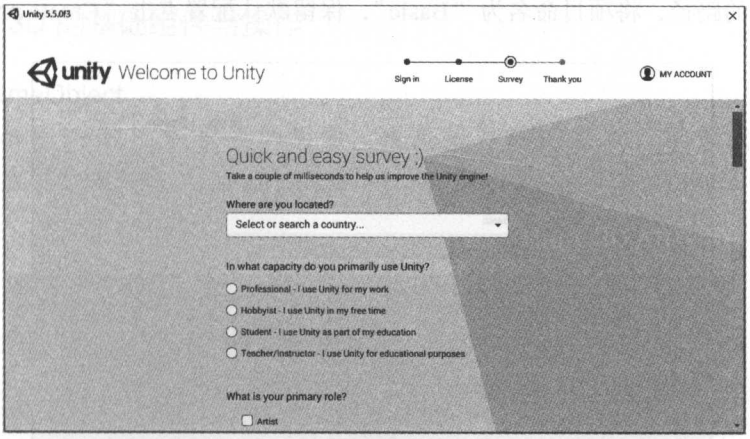
1）双击桌面的 Unity 图标会弹出登录界面，如果没有 Unity 账号可以点击“create one”去官网注册。输入用户名和密码后点击登录。



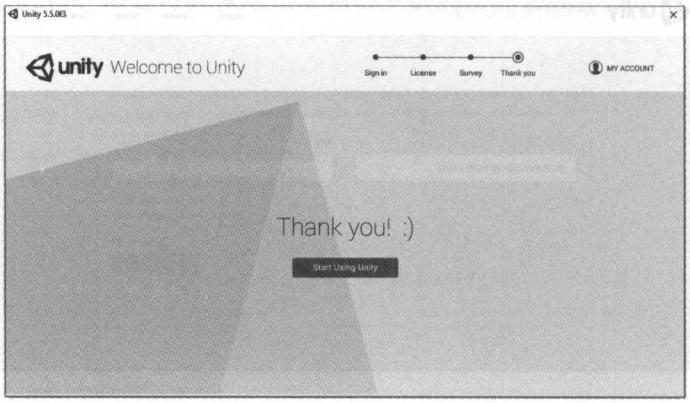
2）等待验证完成后，购买授权的用户输入序列号，个人用户选择 Unity Personal，点击 Next。



3）填写调查问卷，也可以直接点击 OK 跳过，但建议如实填写。

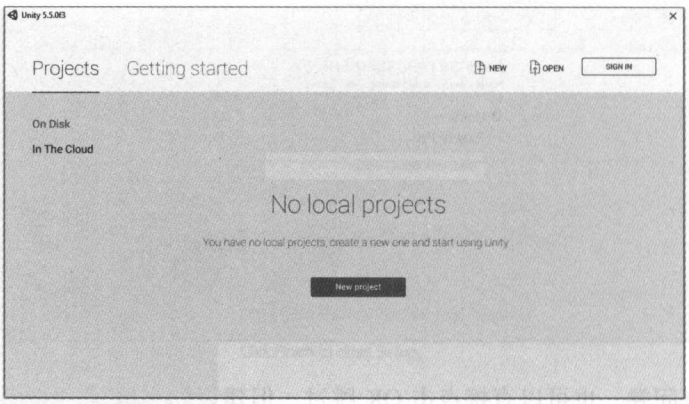


4）点击 Start Using Unity 开始使用。

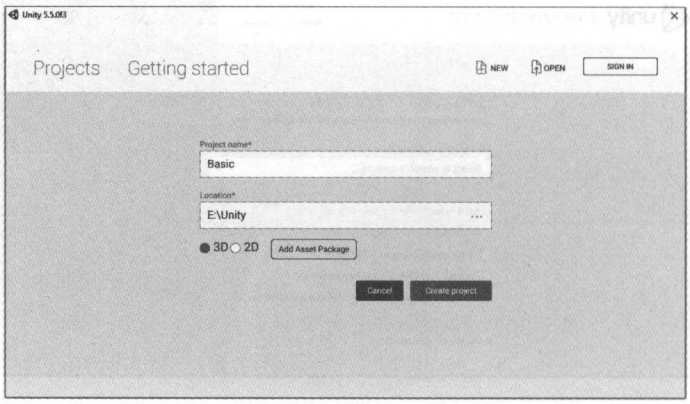


1.1.2 Unity 3D 操作界面简介

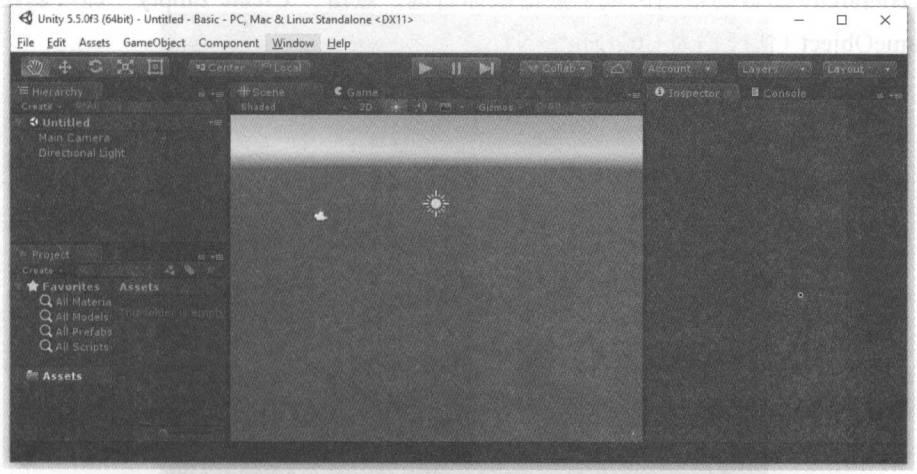
首先我们需要创建一个项目，在项目中介绍 Unity3D 中的基础操作和概念。打开 Unity3D，点击窗口中间的“New project”或窗口右上角的“NEW”创建一个新项目。



选择项目的路径，将项目命名为“Basic”，保留默认配置点击“Create project”。



等待创建默认资源，进入 Unity 主窗口。



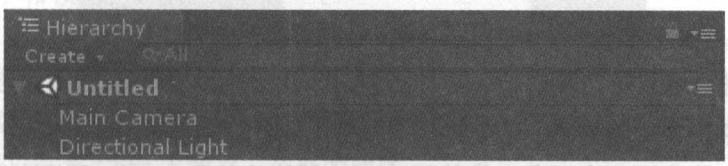
Unity 主窗口的默认布局包括以下几个主要的面板：

- Project：项目中包含的文件，位于 Assets 文件夹下。包括场景（Scene）、模型、贴图、材质、预制件（Prefab）、脚本、动画等。在面板的空白区域中点击右键可以创建指定类型的文件。
- Scene：场景，负责显示目前场景中的可见对象（或编辑器状态下的可见对象）。可以理解为游戏中的一个关卡。可以将一个场景保存为一个场景（Scene）文件，放于 Assets 目录下。
- Hierarchy：当前场景中的 GameObject（场景中的对象）树状图。
- Game：游戏运行面板，点击 Play 按钮后会自动切换至该面板（若面板不存在则会打开）。
- Inspector：监视器面板，显示选中 GameObject 所有组件的属性。
- Console：控制台面板，打印日志 / 警告 / 错误信息。

1.1.3　Unity 3D 的基础组件与操作

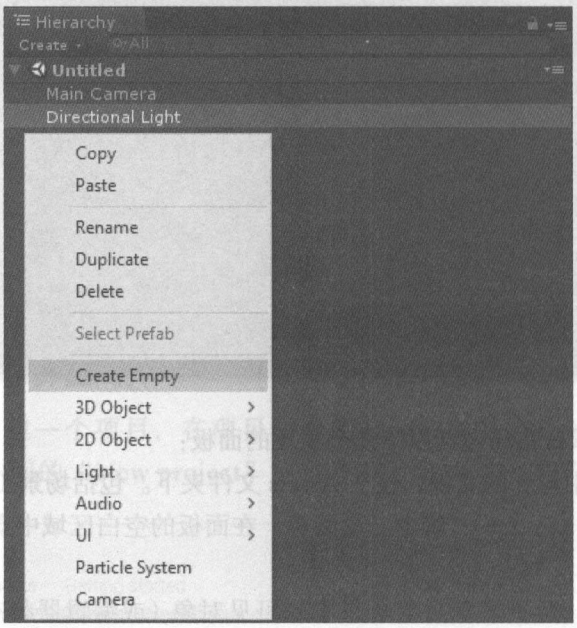
什么是 GameObject

GameObject 是 Unity 中最重要的对象，也是最基础的对象。场景就是由若干个 GameObject 组成的，场景中有且只有 GameObject。例如默认场景中的 Main Camera 和 Directional Light。

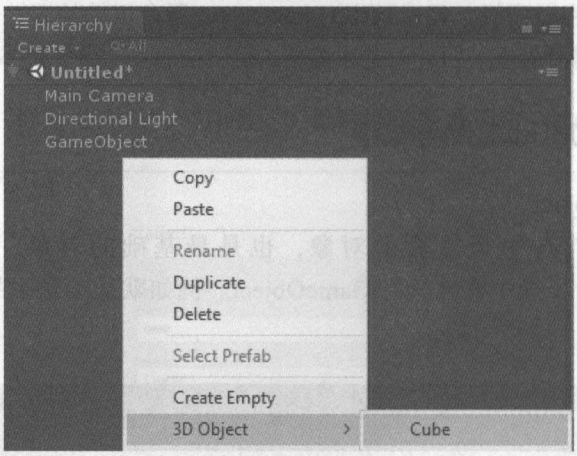


如何创建 GameObject

在 Hierarchy 面板的空白区域上点击鼠标右键，点击“Create Empty”即可创建一个空白的 GameObject（快捷键为 Ctrl+Shift+N）。

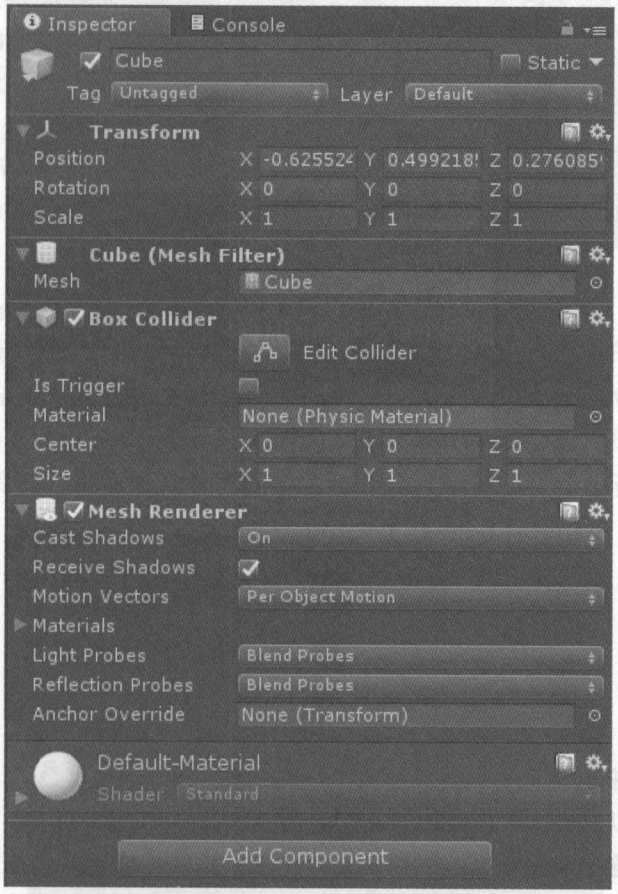


可以注意到在 Scene 面板中看不到任何物体，原因是空白 GameObject 没有任何 Renderer（渲染）组件，因此不会在屏幕上显示。我们可以通过创建一个 Cube 来创建一个可见的 GameObject，在 Hierarchy 的空白处点击鼠标右键，选择 3D Object → Cube 即可创建一个立方体。



可以发现在屏幕中创建了一个立方体（若没有看到，双击在 Hierarchy 中生成的 Cube 可以快速将摄像机聚焦至双击对象上使其可见），左键点击这个 Cube，可以在右侧的 Inspector

面板中查看其所有组件及属性。

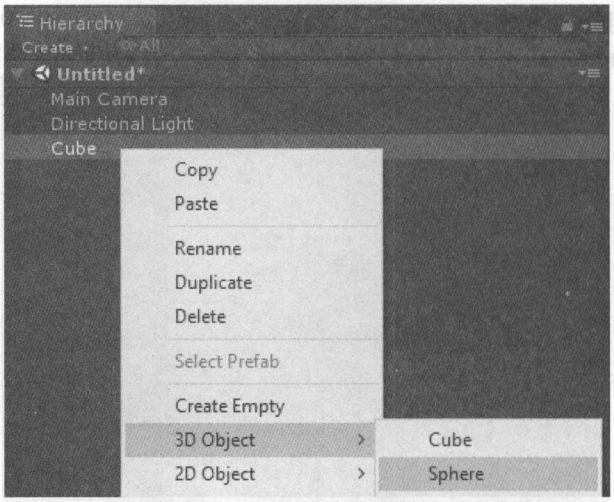


可以发现Cube一共有四个组件：Transform、Mesh Filter、Box Collider、Mesh Renderer。Mesh Renderer是Renderer的一种，因为需要渲染，所以这个立方体在场景中是可见的，Mesh Filter是这个立方体模型文件的引用，Box Collider在后续章节会详细介绍，接下来我们主要看看什么是Transform。

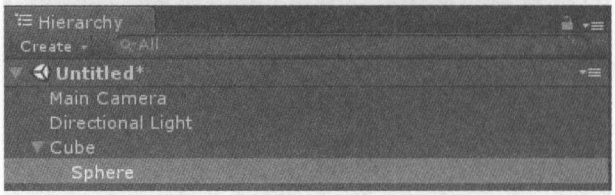
什么是Transform

在Unity中，Transform是用来表示一个GameObject在三维空间中的位置信息和缩放信息的组件，也是Unity的一个核心组件，任何GameObject一定会拥有一个Transform或Transform的子类（RectTransform）。Transform有三个最重要的数据：Position（位置）、Rotation（旋转）和Scale（缩放）。

Transform是树形级联的，也就是一个Transform下可以有多个Transform，就像一个文件夹下可以有多个文件夹一样。因此Hierarchy面板是一个树状图的显示面板。例如我们可以在刚才建立的立方体下再建立一个立方体：在Hierarchy面板的Cube上点击右键，选择3D Object → Sphere在Cube下建立一个新的球体。



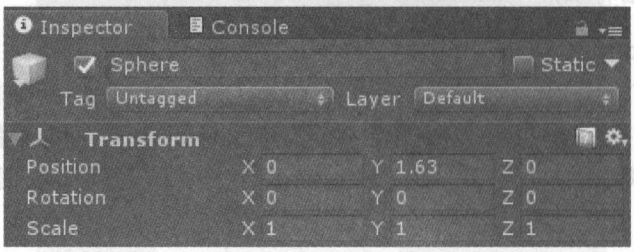
结果如下图所示。



但是场景中并未出现这个球体，原因是创建的球体和 Cube 重叠了。选中这个球体，在 Inspector 面板中可以发现这个球体的 Position（位置）的 XYZ 都为 0。

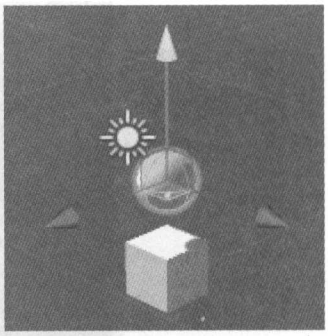
我们可以移动这个球体对象：在 Hierarchy 中选中这个球体，点击左上角工具栏的移动工具（快捷键为 W），在场景中可以发现一个类似坐标的图形，鼠标左键拖曳方向箭头可以移动选中物体，我们将球体移动到立方体的上方。

确定该球体为选中状态（如未选中，可在 Hierarchy 面板中单击选中），这时观察 Inspector 面板中的 Transform 会发现 Position（位置）的 Y 值变为了一个正数。



这里 Transform 显示的值是相对于该对象的上一级对象（父节点）的位置，也就是说球体（Sphere）相对于立方体（Cube）高了 1.63。我们可以选中 Cube，将它移动到其他位置，选中 Sphere 观察它的位置数据还是没有变化。因此，在 Transform 中，所有的数据都是相对

于父节点的偏移量。



如何在场景中移动观察位置

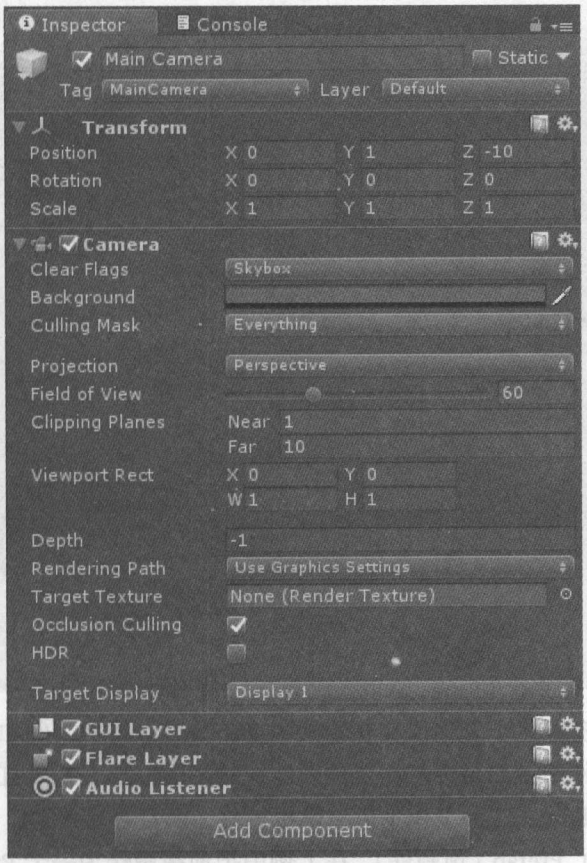
移动：在左上角工具栏选择移动观察位置工具（快捷键 Q），在场景中按住左键拖动屏幕即可移动；或直接按住鼠标中键移动。

缩放：鼠标滚轮。

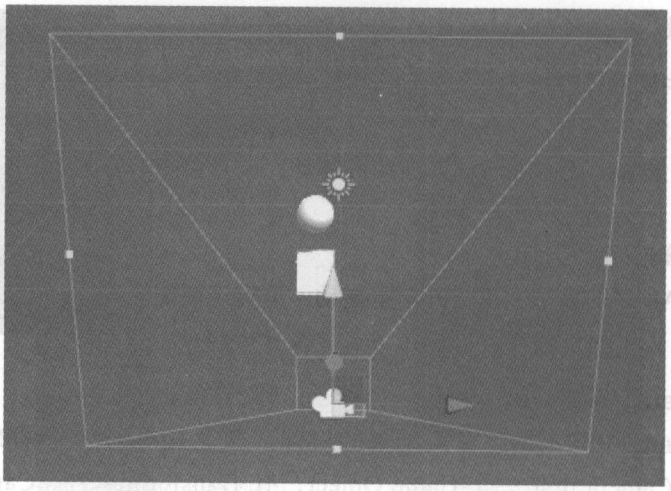
旋转：按住 Alt 键加鼠标左键在屏幕中拖动，可实现视图的旋转。

什么是 Camera

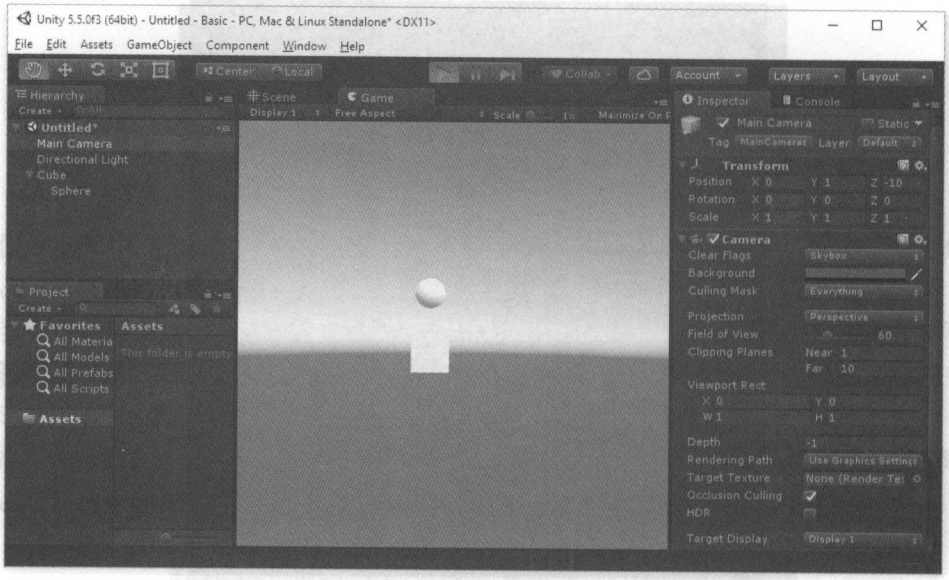
Camera 是 Unity 中显示最终渲染结果的组件，选中 Hierarchy 中的 Main Camera，在 Inspector 中观察 Camera 也是一个 Game Object，有 Transform 组件，Camera 本身也是一个组件，Hierarchy 的 Main Camera 只是因为拥有 Camera 这个组件所以才有 Camera 的功能，否则它也只是一个普通的 GameObject。



如上图调整 Camera 的 Near 和 Far 的值，选中后可以观察其视锥的可视区域，仅在该区域内的物体的渲染结果才会在屏幕上显示。



调整 Cube 的位置，使其处于 Main Camera 的视锥内，点击 Scene 面板上方的 Play 按钮 可以显示最终的渲染结果。



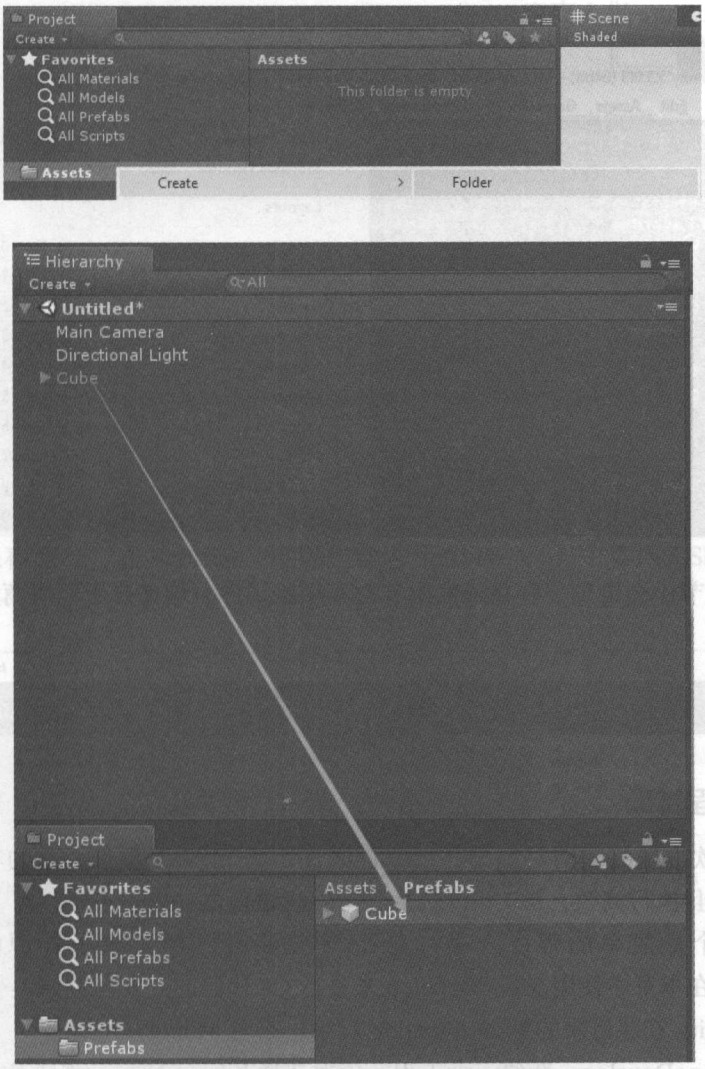
什么是 Prefab

在项目的制作过程中，往往需要多次使用同一个定义好的资源。例如游戏中的地板、门、机关等。将这些资源逐个复制保存在场景中会十分难以维护。例如有 100 个地板分别位于 3 个不同的场景文件中，如果突然需要更新地板的贴图，那么就要打开这 3 个场景，逐个更新这 100 个地板的贴图。这种工作不但枯燥机械化，而且容易出错。

因此 Unity 推出了 Prefab（预制件），在上例中把地板制作为一个 Prefab，在需要这个地板的场景中使用这个 Prefab。当需要更新贴图时，直接更新这个 Prefab，使用这个 Prefab 的场景中的地板的贴图就可以全部被更新。

如何创建 Prefab

Unity 使得创建 Prefab 的过程极其简单且轻松，只需要将在 Hierarchy 面板中配置好的 GameObject 拖至 Project 面板的指定文件夹即可。例如下图将刚才制作的 Cube 直接拖拽至 Assets/Prefabs（在 Project 面板的 Assets 文件夹上点击鼠标右键，选择 Create → Folder 即可创建文件夹）目录下即成功创建了名为 Cube 的 Prefab。



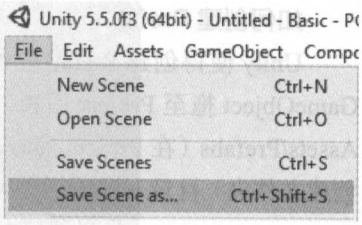
注意 在 Hierarchy 面板中 Prefab 的引用显示为蓝色。

如何保存场景

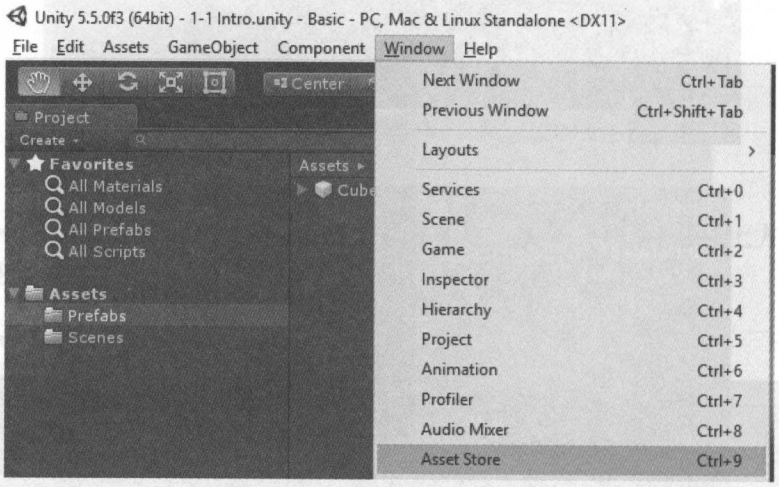
在菜单栏中点击 File → Save Scene as，然后选择路径和文件名即可保存场景。

在本例中，将该场景保存为 Assets/Scenes/1-1 Intro。

1.1.4 Unity 3D 的资源商店



Unity 是一个用户活跃度非常高的群体，有很多游戏制作人会将他们开发或创作的资源上传至 Unity 资源商店，既可以获取一定的收益又可以方便其他开发者。在资源商店中有很多实用的工具和资源。可以从菜单栏的 Window → Asset Store 进入商店。



在之后的章节中会讲到一些在开发中非常实用的插件，均来自于资源商店。

1.2 Unity 3D 编程基础

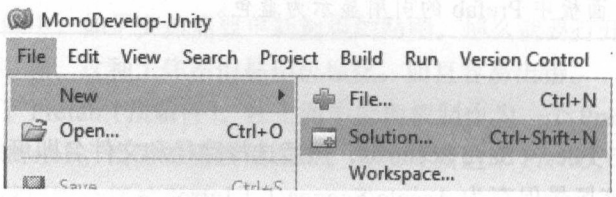
1.2.1 C# 编程基础

C# 是由微软推出的面向对象的高级编程语言，基于 .NET 框架。它的功能强大，语法严谨，设计精良且执行高效，被 Unity 选为脚本编程语言之一。

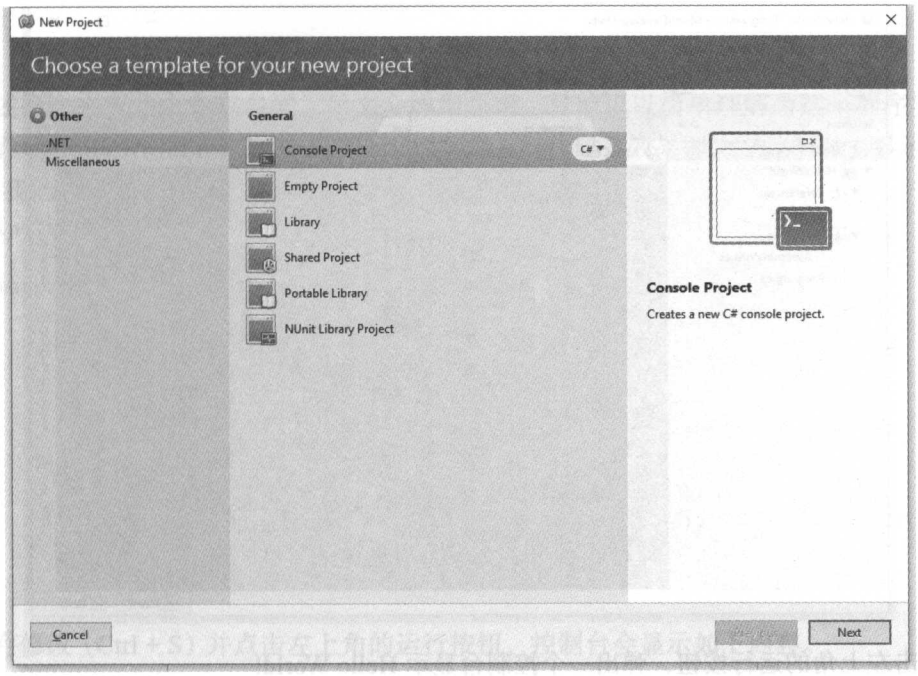
首先从一个最简单的程序入手，在 Unity 安装时已经集成了开源的 C# 开发环境 MonoDevelop，在本章中使用 MonoDevelop 来进行 C# 代码的开发与调试。

1）进入 Unity 安装目录的 MonoDevelop 目录，启动 MonoDevelop。

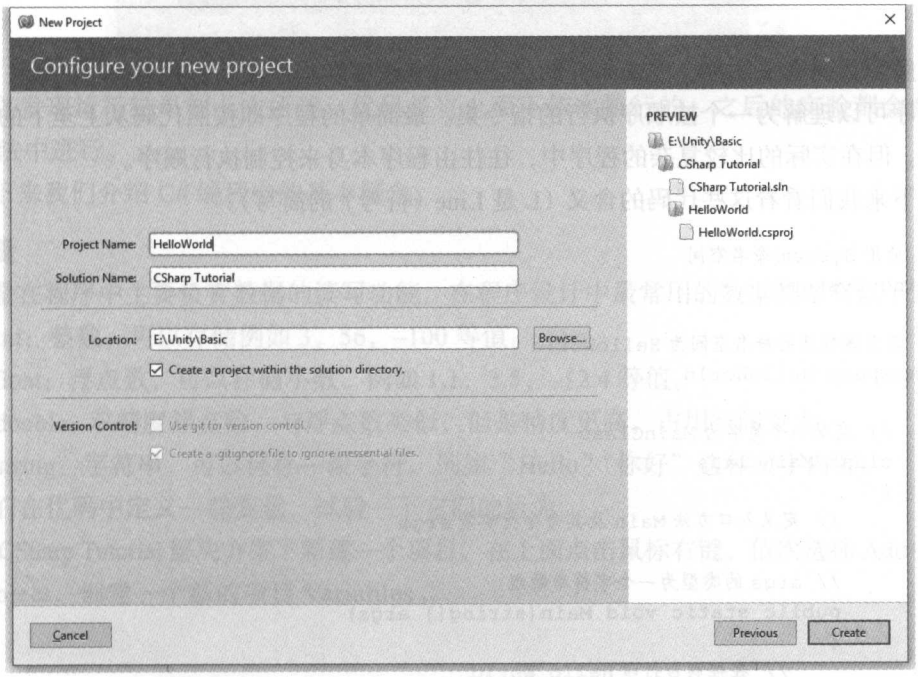
2）打开 MonoDevelop，新建一个工程，依次选择 File → New → Solution。



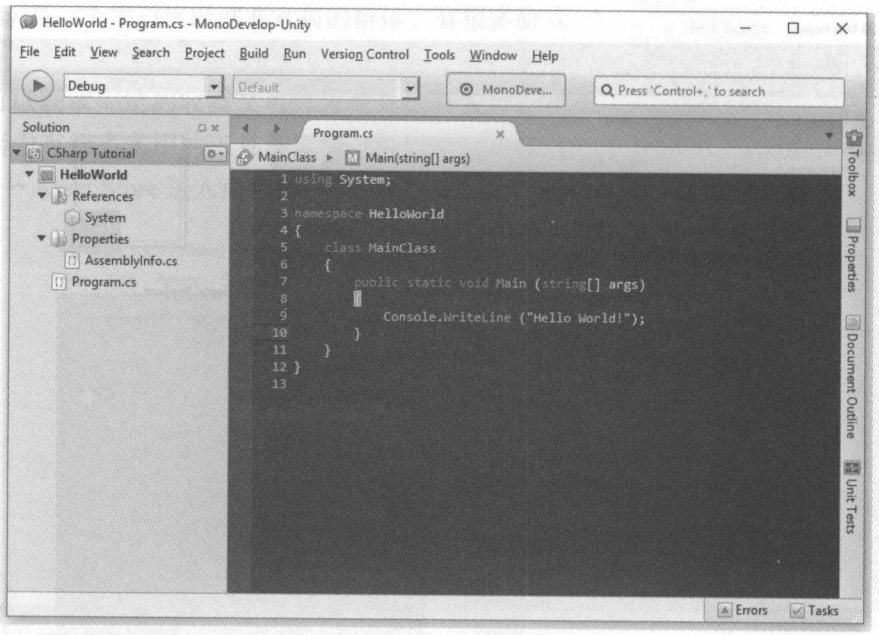
3）选择 Console Project，点击 Next。



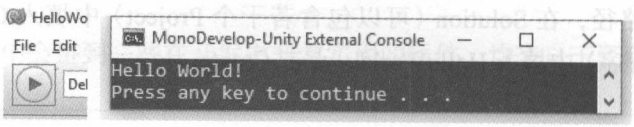
4）选择保存路径，在 Solution（可以包含若干个 Project）中填入 CSharp Tutorial，在 Project（对应一个程序）中填入 HelloWorld，点击 Create。



可以发现 MonoDevelop 为我们创建了如下目录，并在 Program.cs 文件中添加了代码。



点击左上角的运行按钮，弹出一个控制台显示 Hello World!



在解释代码含义之前，先简单介绍一下什么是程序和程序的运行机制。

程序可以理解为一个按顺序执行的指令集。最简单的程序即按照代码从上至下的顺序依次执行。但在实际的比较复杂的程序中，往往由程序本身来控制执行顺序。

接下来我们看看这些代码的含义（L 是 Line（行号）的简写）。

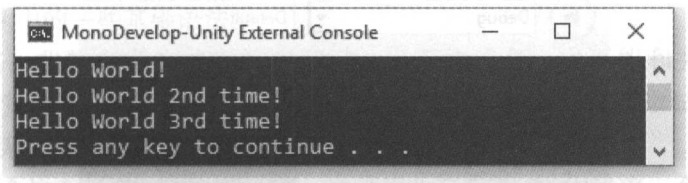
```
// 使用 System 命名空间
using System;

// 定义该程序的命名空间为 HelloWorld
namespace HelloWorld
{
    // 定义一个类名为 MainClass
    class MainClass
    {
        // 定义入口方法 Main 及其命令行参数 args

        // args 的类型为一个字符串数组
        public static void Main(string[] args)
        {
            // 在控制台打印 Hello World
```

```
            Console.WriteLine("Hello World!");
        }
    }
}
```

上述概念中有几个专有名词，后文会详细介绍。目前可以简单理解为这个程序实质上仅执行了 Console.WriteLine（“Hello World!”）这一行代码，效果即为在控制台上显示 Hello World。我们可以在该代码后追加以下代码，使程序变为：

```
using System;

namespace HelloWorld
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("Hello World 2nd time!");
            Console.WriteLine("Hello World 3rd time!");
        }
    }
}
```

保存修改（Ctrl + S）并点击左上角的运行按钮，控制台会显示如下内容。



可以发现运行结果如上文所述，代码是自上而下依次执行的。之后的实验都会在这个 Main 方法中进行。

接下来我们介绍 C# 编程中的基本概念。

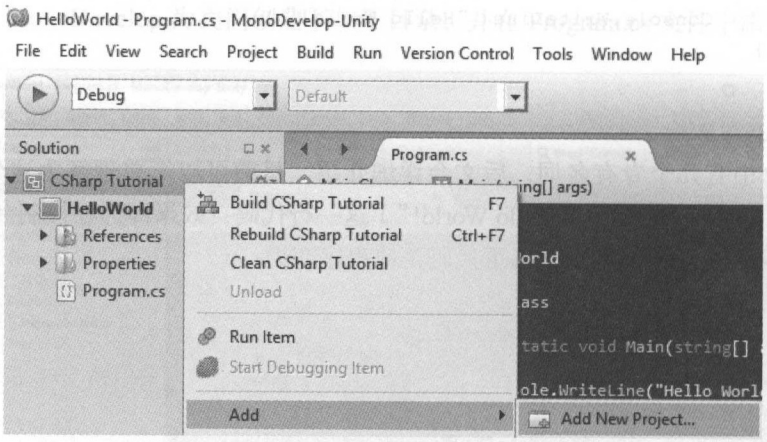
变量

变量在程序中主要负责数据的读写功能。在程序设计中最常用的数据类型有以下几种：

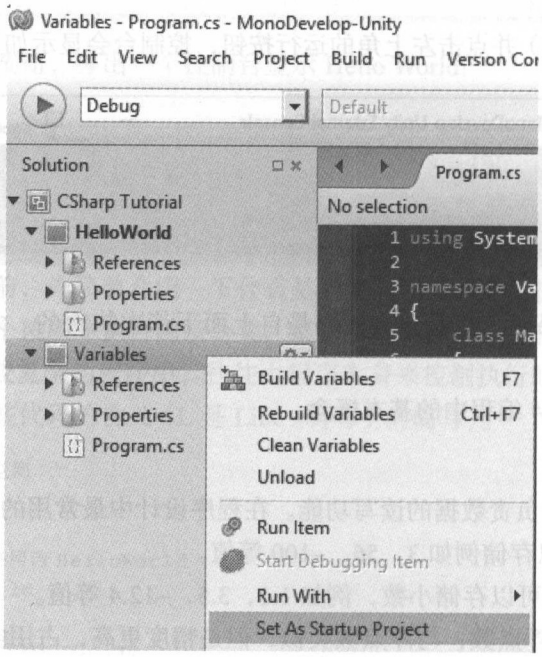
- int：整数，可以存储例如 3，56，–100 等值。
- float：浮点数，可以存储小数，例如 1.1，3.5，–12.4 等值。
- double：双精度浮点数，与浮点数类似，但是精度更高，占用空间更大。
- string：字符串，可以保存一段字符，例如“Hello”“你好”这种文字内容。

我们在代码中定义一些变量，试验一下它们的行为。

在 CSharp Tutorial 解决方案下新建一个项目，在上面点击鼠标右键，依次选择 Add → Add New Project，创建一个新的项目 Variables。



现在解决方案 CSharp Tutorial 中存在两个项目，每个项目对应一个可执行程序，点击右上角运行时编辑器启动的还是“Hello World”这个项目对应的程序，但是我们现在需要调试运行的是新建的 Variables 项目，因此，需要把该项目设为移动项目：在 Variables 上点击鼠标右键选择 Set As Startup Project。



我们将生成的 Program.cs 中的代码改为如下代码：

```
using System;

namespace Variables
{
    class MainClass
```

```
    {
        public static void Main(string[] args)
        {
            int   testInt = 1;
            float testFloat;
            testFloat = 0.5f;
            string testString = "Hello World by string" ;
            Console.WriteLine(testInt);
            Console.WriteLine(testFloat);
            Console.WriteLine(testString);
        }
    }
}
```

我们来观察 int testInt = 1，这行代码的意义是定义了一个名为 testInt 的整型变量，并将其初始值设为 1。后面的代码意义类似，定义了一个名为 testFloat 的浮点型变量和名为 testString 的字符串变量。

值得注意的是变量的内容可以用“=”来修改，这里的等号并不是数学里的判断是否相等，而是把等号右边的值复制到等号左边，也就是将内容写入等号左边的变量中。例如“testFloat = 0.5f;”，这里就是把 testFloat 的值更新为 0.5，后面的 f 代表这是一个浮点数，不写的话系统默认会认为是一个双精度类型，将高级别的数据赋值给低级别的数据在 C# 中会被认为是一种错误，编译器会报错。

这里介绍 C# 中的一些基础语法。

变量的命名：非数字开头的字符串即可，但不能重名，也不能和关键字（例如 int、string、void 等）重名（大小写敏感，例如 test 和 Test 是不重名的）。可以试想如果重名，在运行时编译器将无法判断使用的到底是哪一个变量。

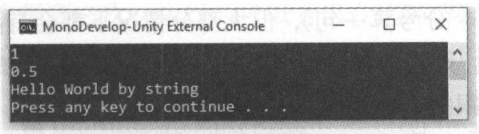
分号：大家会注意到每句以分号结尾，这是因为分号在 C# 中代表一个操作的结束。编译器遇到分号时便明白这一句结束了。

空格：从语法上来说，空格仅有分割内容的作用，具体有几个空格其实无所谓，例如：

```
float testFloat;
float          testFloat;
```

在编译器看来都是一样的，另外等号左右的空格并不是必要的。

运行这个程序，显示如下结果：



可以看出程序依然是顺序执行，依次打印出了变量内的内容。

条件控制语句

在 C# 程序中，当需要进行逻辑判断时，会用到条件判断语句。在自然语言中即：如果

满足条件 A，执行指令 A1，如果满足条件 B，执行指令 B1。

if：顾名思义，如果怎么样，则执行 if 语句块中的内容。

else：与 if 连用（不是必须），当与其连用的 if 中的条件不满足，则执行 else 语句块的内容。可以直译为“否则”。

switch：开关语句，与 case 搭配使用，具体参考以下例子。

新建一个名为 Condition 的项目并将其设为启动项目，改写 Main 方法：

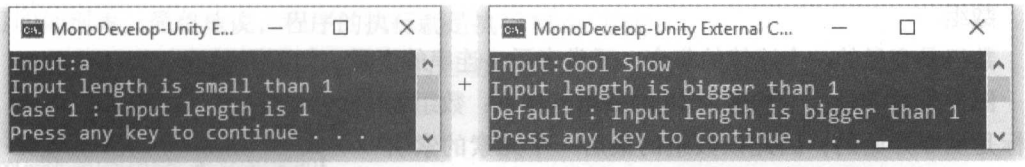
```
public static void Main(string[] args)
{
    Console.Write( “Input:” );
    string input = Console.ReadLine();
    if (input.Length > 1) {
        Console.WriteLine( “Input length is bigger than 1” );
    } else {
        Console.WriteLine( “Input length is small than 1” );
    }

    switch (input.Length)
    {
        case 0:
            Console.WriteLine( “Case 0 : Input length is 0” );
            break;
        case 1:
            Console.WriteLine( “Case 1 : Input length is 1” );
            break;
        default:
            Console.WriteLine( “Default : Input length is bigger than 1” );
            break;
    }
}
```

首先在控制台打印提示语句“Input:”，但并不换行，这就是 Console.Write 和 Console.WriteLine 的区别。string input = Console.ReadLine() 是说从控制台输入一个字符串，将其内容存到名为 input 的字符串变量中，接下来在 if 中判断 input（也就是刚才输入的字符串）的长度是否大于 1，input.Length 可以获取这个字符串的长度。结果是一个整数。如果满足条件，则打印“Input length is bigger than 1”，否则打印“Input length is small than 1”，也就是执行 else 中的内容。

花括号（大括号）中的内容就是一个语句块，通常表示一段需要一起执行的代码内容（本例中虽然只有一句（一个分号算一句），但大部分情况下都会有两句以上）。如果在满足 if 条件时需要执行多个语句，则需要把该语句放在 if 的花括号内。如果没有花括号，编译器会认为 if 语句只包含其后紧跟的一句代码。

switch 中则是具体的条件，例如 case 0 的意思是说当 switch 中的参数等于 0 时，执行 case 下的语句。注意在每个 case 后需要添加 break，否则会继续执行代码中下一个 case 中的代码，这种特性有时候会需要用到。

执行程序，尝试输入不同的内容，观察运行结果。



循环语句

在程序应用中，往往需要循环执行一些操作来满足特定的功能需求。C# 中主要有以下两种循环控制语句。

while：满足条件则循环执行。

for：满足条件则循环执行，通常用于确定循环次数的场合，语法中包括三部分：

- 初始条件
- 循环条件
- 步长（每次循环都会进行的一个操作）

在代码中详解，首先创建项目 Loop 并设为启动项目，将 Main 方法改为：

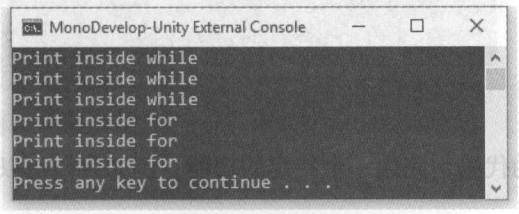
```
public static void Main(string[] args)
{
    int counter = 0;
    while (counter < 3) {
        Console.WriteLine("Print inside while");
        counter++;
    }

    for (int i = 0; i < 3; i++) {
        Console.WriteLine("Print inside for");
    }
}
```

首先定义了一个整型变量 counter 并赋值为 0，然后在 while 中判断如果 counter 小于 3 则循环 while 语句块中的内容。在 while 中 counter++ 意思为 counter 加 1。因此这段循环会执行三次。

在 for 循环中上文提到的三部分在代码中对应的内容为：

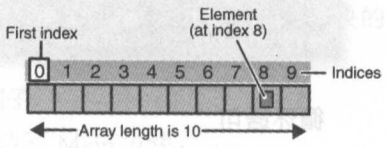
- 初始条件：定义了一个整型变量 i，并且将 i 的初始值设为 0。
- 循环条件：i 的值小于 3。
- 步长：每次循环为 i 加 1，相当于在 while 中的 counter++。

因此这个 for 循环也会循环三次，运行后可以看出在这种情况下这两个循环其实是等效的，实际应用中其实也可以相互替换。但是在一些特定情况下使用其中一个会更便捷。



数组

数组是变量的一个连续的集合，通常表示一连串的变量，如下图所示：



图中所示的是一个长度为 10 的数组，注意：数组的下标是从 0 开始的，因此该数组的最后一个元素的索引是 9。实际使用过程中一定要注意。

在 C# 中，我们可以用以下方式定义一个字符串数组：

```
string[] texts = new string[3];
```

这里的方括号代表 texts 是一个数组，类型是 string，3 代表这个数组共有 3 个元素。具体用法可以参考以下应用实例。

首先新建一个名为 Array 的项目并设为启动项目，修改 Main 方法。

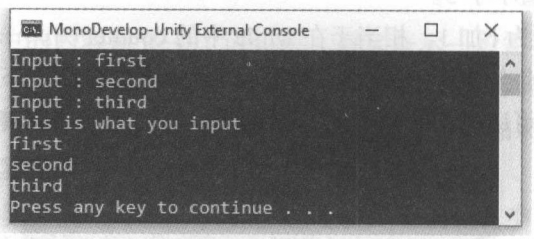
```
public static void Main(string[] args)
{
    string[] texts = new string[3];
    for (int i = 0; i < texts.Length; i++) {
        Console.Write( “Input : “);
        texts[i] = Console.ReadLine();
    }

    Console.WriteLine( “This is what you input” );
    for (int i = 0; i < texts.Length; i++) {
        Console.WriteLine(texts[i]);
    }
}
```

我们定义了一个字符串数组 texts，然后使用循环按顺序从控制台读取输入，texts.Length 可以取得数组的元素个数，在循环中要确保数组的下标小于数组的长度，也就是下标不会超过 2，这样可以保证数组的访问不会越界（超过数组的长度）。

循环内部首先打印了提示语句“Input：”，接下来在读取输入时将输入的字符串赋予了 text[i]，text[i] 即代表第 i 个数组中的元素，循环执行中，i 会从 0 一直循环至 2，也就是循环三次。依次记录输入的内容。

后面的一个循环打印出刚才记录的输入内容，也是按照 0,1,2 的顺序打印的。

运行，输入 3 次字符串观察结果。



方法

方法是一段打包的代码，完成一个特定的功能。例如我们可以实现一个加法处理，或批

量打印处理等。简单地说，程序的执行就是执行 Main 方法。

新建一个名为 Method 的项目并设为启动项目，将 Program.cs 改为如下内容：

```
using System;

namespace Method
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine(Add(1, 2));
        }

        static int Add(int a, int b)
        {
            int sum = a + b;
            return sum;
        }
    }
}
```

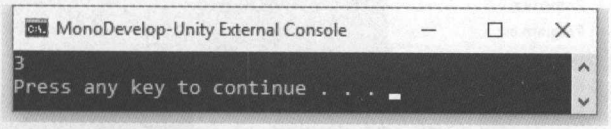
首先详细解读一下 Add 方法：

```
static int Add(int a, int b)
```

第一行也就是 Add 方法的定义，static 后续章节中有详细介绍，int 为返回值的类型，即这个方法执行之后在调用方法的地方返回一个整数。Add 为方法的名字，命名规则与变量相同，不能重名，不能以数字开头。小括号内的是参数，这个方法有两个输入参数，以逗号隔开。是名为 a 和 b 的整数。方法内部代码需要在花括号内。在方法中将 a 和 b 的结果赋予了新的变量 sum，然后将 sum 作为返回值返回。

在 Main 方法中调用了 Add 方法，参数 a 传入为 1，b 传入 2。Add 执行完毕后的返回值就是返回到了这里，最终被打印在控制台上。

运行这个程序，得到以下结果：



其实 Main 方法也是一个方法，但是它是一个特殊的方法，简单理解就是 Main 方法就是为了告诉编译器这个程序从哪里开始执行。Main 方法的参数就是命令行参数，命令行参数就是执行一个程序时传入的参数。

举个简单的例子，可以在 D 盘根目录建立一个文本文档 test.txt，在其中随便输入一段内容，在 Windows 的运行窗口中输入以下命令：

```
notepad D:\\test.txt
```

回车执行可以发现记事本打开了刚才在 D 盘创建的文件。这里 notepad 后面的 D:\\test.txt 就是命令行参数，记事本程序在它的 Main 方法中获取这个参数后便执行了默认操作：打开这个文件。命令行参数就是这样从外部程序获取一些启动数据的。

1.2.2 C# 面向对象编程基础

类

在面向对象编程中，类是最重要的一个概念。可以简单理解为封装了方法和变量的一个代码集合。通常是有关联的一个数据及其操作的集合。例如一个位置类 Position，有数据 x 和 y 代表它的坐标，有 CalcDistanceFrom 方法可以计算这个点到另一个点之间的距离。

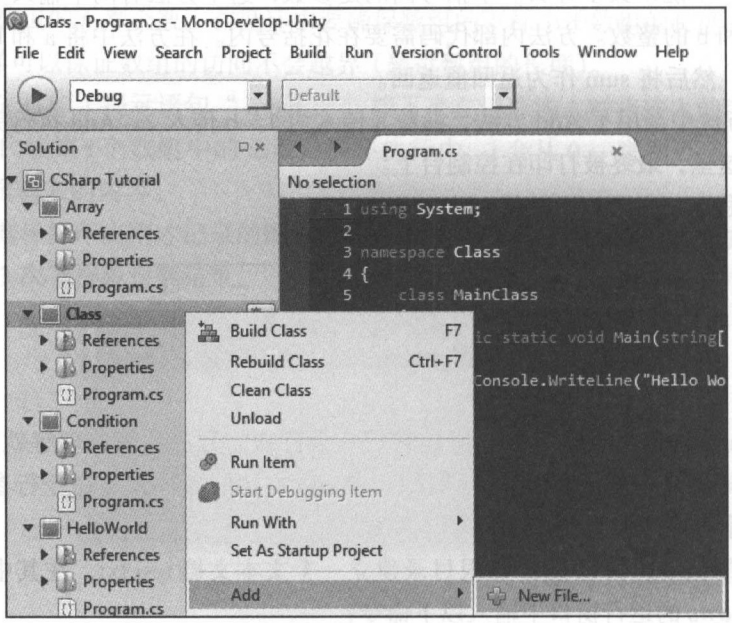
对象

对象是类的实例。这句经典的翻译虽然存在很久但是从表达上来说非常晦涩难懂。举例说明：手机是一个类，而我的手机则是一个对象。手机是一个很宽泛的概念，但是我的手机则是一台十分具体的手机。它真实存在。Position 类是一个代表位置的类，而 A 点（0,1）是 Position 类的一个对象。因为 A 点是一个具体的实例。

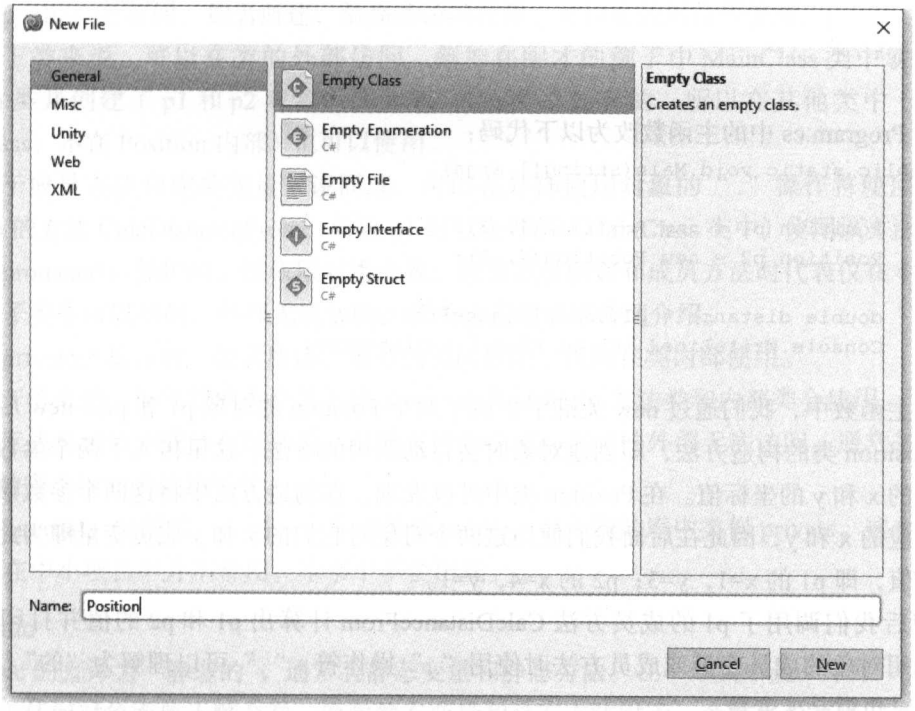
构造方法

在生成对象时会自动调用的方法，一般用于初始化。例如创建一个 Position 对象时可以在构造方法中传入它的坐标，在构造方法中自动给成员变量坐标赋值。

下面写一个项目来了解这些概念的意义：新建一个名为 Class 的项目并设为启动项目。创建一个新的文件：在 Class 工程上右击，依次选择 Add → New File。



确认选择为 General → Empty Class，将其命名为 Position，点击 New。



在 Position.cs 中添加以下代码：

```
using System;

namespace Class
{
    public class Position
    {
        public double x;
        public double y;

        public Position()
        {
            x = 0;
            y = 0;
        }

        public Position(double x, double y)
        {
            this.x = x;
            this.y = y;
        }

        public double CalcDistanceFrom(Position p)
        {
            double deltaX = x - p.x;
            double deltaY = y - p.y;
            return Math.Sqrt(deltaX * deltaX + deltaY * deltaY);
```

```
        }
    }
}
```

将 Program.cs 中的主函数改为以下代码：

```
public static void Main(string[] args)
{
    Position p1 = new Position(1, 3);
    Position p2 = new Position(4, 1);

    double distance = p1.CalcDistanceFrom(p2);
    Console.WriteLine("P1 to P2 : " + distance);
}
```

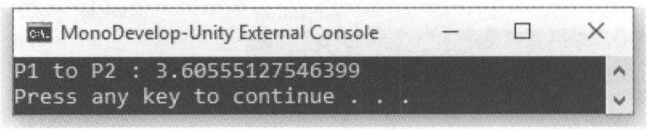
在主函数中，我们通过 new 关键字生成了两个 Position 的对象 p1 和 p2。new 后面的方法即 Position 类的构造方法，即创建对象时会自动调用的方法。这里传入了两个参数分别为两个点的 x 和 y 的坐标值。在 Position 类中可以发现，在构造方法中将这两个参数赋予了该对象对应的 x 和 y，因此在后面我们使用这两个对象时它们的 x 和 y 成员变量即为此处传入的参数值，即 p1 的 x=1，y=3；p2 的 x=4，y=1。

随后我们调用了 p1 的成员方法 CalcDistanceFrom 计算出 p1 和 p2 的值并打印到屏幕上。使用对象的成员变量或成员方法时使用“.”操作符。“.”可以理解为“的”。p1.x 即 p1 的 x，也就是 x 变量。

Position 类正如上述的例子所示：每个对象有两个成员变量 x 和 y。有一个成员函数 CalcDistanceFrom 可以计算该对象与另外一个 Position 对象之间的距离。可以注意到两个 Position 函数是没有返回值的，因为它们是构造方法。Main 方法中的 p1 和 p2 是含有两个输入参数的构造方法创建的，因此在该函数执行后 x 和 y 为输入参数对应的值。

注意在有参数的构造方法中出现了 this，意思为“此对象的”。主要用于参数名和成员变量同名的场合。

运行程序，结果如下：



1.2.3 C# 面向对象编程进阶

访问权限

在上例中可以发现 Position 类名前，成员变量和成员方法前都有一个 public 关键字，这是描述访问权限的关键字，访问权限可简单理解为能不能使用。

对于类来说就是这个类能不能在外部（也就是这个类代码花括号之外的区域）使用。

对于类中的成员变量或成员方法来说就是能不能被该类的对象用“.”操作符使用。

C# 中的访问权限有以下几种：

- public：公有的。如名所述，最高的访问权限，可以在任何位置使用。

对于类来说，可以在类的外部访问。例如在刚才的例子中 MainClass 类中就使用了 Position 类并创建了 p1 和 p2 对象，因为 Position 类是公有的，所以在其他类中（这里即 MainClass，不在 Position 内部）也可以使用。

对于成员方法和成员变量来说同理，可以在外部使用对象的“.”操作符使用。例如 Position 的方法 CalcDistanceFrom，对象 p1 可以在外部（MainClass 类中）使用该方法。

- protected：保护的。该权限对类无效，修饰成员函数和成员方法时代表仅在本类及其子类中可以访问，外部无法访问。子类会在继承中详细介绍。
- private：私有的。如名所述，最低的访问权限。仅能在类内部使用。

对于类来说，即外部的类全都不能访问。通常仅 Main 方法类和内部类会使用。

对于成员函数和成员变量来说，只能在内部访问。在类的外部无法访问。通常是为了保证数据的安全性和类的封装性。

- internal：内部的。在同一个库中类似 public，在不同的库中类似 private。这个关键词在中小项目中并不常用，本书中不会出现。

static

static 的直译为“静态的”，通常有静态变量和静态方法。在一个类中是一个唯一的存在，它不属于任何对象而属于整个类。通常静态变量和静态方法对于一个类来说有唯一性和特殊性。

通过以下例子详细讲述 private 和 static 配合使用的场合和特性，将上例中的 Position 类修改为如下代码。

```
using System;

namespace Class
{
    public class Position
    {
        public double x;
        public double y;

        private int index;

        private static int instanceCount;

        public Position()
        {
            x = 0;
            y = 0;
            UpdateIndex();
        }

        public Position(double x, double y)
        {
            this.x = x;
```

```
            this.y = y;
            UpdateIndex();
        }

        public double CalcDistanceFrom(Position p)
        {
            double deltaX = x - p.x;
            double deltaY = y - p.y;
            return Math.Sqrt(deltaX * deltaX + deltaY * deltaY);
        }

        private void UpdateIndex()
        {
            index = instanceCount;
            instanceCount++;
        }

        public int GetIndex()
        {
            return index;
        }
    }
}
```

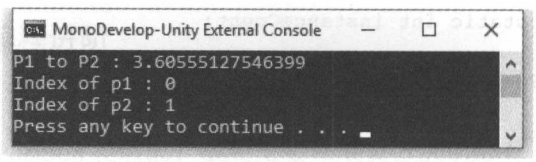
可以看到添加了一个私有的成员变量 index 来表示该对象序号，即该对象是第几个生成的。添加了一个私有的静态变量 instanceCount，用来统计对象的数量，即总共生成了多少个对象。在构造函数中调用一个名为 UpdateIndex 的函数，每次将 index 赋值为现有的对象数，并为 instanceCount 加 1。

因为静态变量在类中是唯一的，所以每次在 UpdateIndex 中调用的 instanceCount 都是同一个变量，而局部变量 index 是每个对象拥有一个，这是一个典型的静态变量的应用场景，通过静态变量和成员变量的结合使用完成序号更新和对象数量统计的功能。

在刚才的 Main 函数中添加以下两行代码：

```
Console.WriteLine("Index of p1 : " + p1.GetIndex());
Console.WriteLine("Index of p2 : " + p2.GetIndex());
```

运行并观察控制台输出结果。



可以发现如预期的一样，p1 和 p2 被更新了。

将 index 和 instanceCount 设为 private，主要是为了写保护，也就是我并不希望在外部改动这两个值。因为当这两个值在外部被更新时，响应的功能会发生混乱：序号可能重复或者对象总数统计错误。例如如果在 Main 方法的结尾处添加如下代码：

```
p1.index = 3;
Position.instanceCount = 0;
```

这时 p1 的序号错误，而且统计数也是错的。为了防止这种类的使用者的误操作，所以将这两个变量设为私有的，这样就算有人想更改这些数据，在编译时也会引发编译器错误，提示他这两个变量无法访问。这就是 private 的封装性和安全性。

可以创建了一个名为 GetIndex 的公有成员方法获取 index，这样可以保证外部对 index 的使用是可读且只读的。

继承

继承是类和类之间的一种关系，如果说 A 类继承了 B 类，则 A 类的对象拥有 B 类中的所有成员变量和成员方法。在面向对象编程中是一种代码复用的机制。在 C# 中，所有的类如果不明确说明继承关系，编译器都默认将其继承于 Object。

自然逻辑的继承其实很简单，例如如果游戏是一个类，那么 FPS 游戏、MOBA 游戏和沙盒游戏都继承了游戏，是游戏的一个子类，游戏是它们的父类（也叫基类）。它们有游戏的所有特点（基类的成员变量和成员方法）而且还有自己独特的额外特点（子类本身的成员变量和成员方法）。

例如我们可以创建一个类 TaggedPosition 继承于 Position，但是这是拥有标识的 Position。创建 TaggedPosition.cs 并添加以下代码：

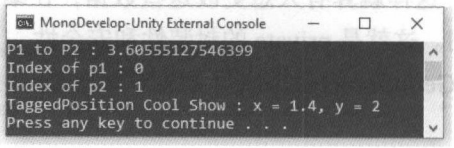
```
using System;

namespace Class
{
    public class TaggedPosition : Position
    {
        public string tag;

        public TaggedPosition(string tag, double x, double y)
            : base(x, y)
        {
            this.tag = tag;
        }

        public override string ToString()
        {
            return "TaggedPosition " + tag + " : x = " + x + ", y = " + y;
        }
    }
}
```

在类名 TaggedPosition 后紧接着一个冒号就是继承的语法，后面则是这个类继承的类。在构造方法下面一行的 base 表示这里需要使用父类的构造方法，通过这一句将传入的坐标赋予本对象。因为继承了 Position，所以 TaggedPosition 对象其实也是有成员变量 x 和 y 的。最后添加了一个 ToString 方法来返回这个类，打印时显示字符串。

在 Main 方法中添加如下代码：

```
TaggedPosition tp = new TaggedPosition( "Cool Show" , 1.4, 2.0);
Console.WriteLine(tp.ToString());
```

运行程序：



命名空间

在使用别人开发的代码（或库）时，往往会出现命名重名的时候。例如开发过程中定义了一个名为 Position 的类，使用他人代码时也有可能这个人也定义了一个 Position 的类。这时命名冲突会导致错误。因此 C# 引入了命名空间的概念，顾名思义就是一个命名的空间，不同命名空间的同名类编译器可以通过命名空间来区分避免错误。另一方面也会使代码更加整洁，将不同模块的代码区分开。而且在开发大型项目时可以更有效地组织模块间的依赖关系。

可以发现在刚才的代码文件中，每个 cs 文件的第一行都是 using System;，这句的意思是使用 System 这个命名空间，其实 Console 类就在 System 命名空间中。WriteLine 是 Console 的一个公有静态方法，因为有这行 using 我们才可以像这样使用 Console 类。每个类名的前一行都是 namespace Class，这里的 Class 就是我们这个程序的命名空间，相同命名空间下的类互相引用是不需要添加 using 的，所以可以在 Main 方法中使用 Position 类和 TaggedPosition 类。

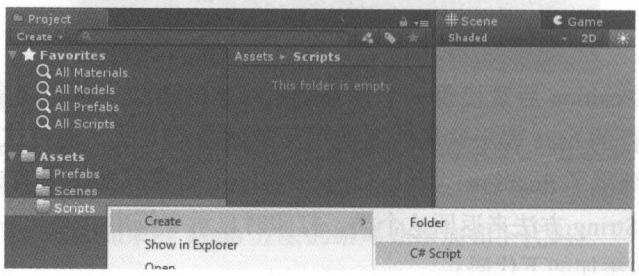
1.2.4 Unity 3D 中的 C# 脚本

在 Unity3D 中，C# 是用于控制游戏逻辑的脚本控制语言。脚本在程序运行中并不是必须的，因此游戏程序本身的 Main 函数是不需要游戏逻辑程序控制的，Unity 已经写好。我们需要编写 Unity 提供一些特定事件的处理逻辑。

Unity 引擎的类库功能丰富且强大，同时也非常庞大。但是入门很简单，我们以 MonoBehavior 类入手。事实上，实际应用中脚本编写的入口就是这个类。我们通过一个例子了解在 Unity 中如何使用 C# 脚本。

1）打开在 1.1 书中创建的项目，在 Assets/Scenes 目录下新建一个名为“1-2 Basic Script”的场景。

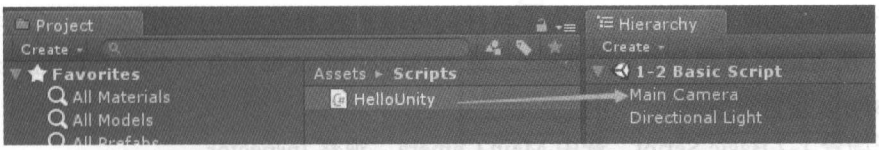
2）在 Assets 目录下创建文件夹 Scripts，在该目录点击鼠标右键，依次选择 Create → C# Script，创建一个 C# 脚本文件，并命名为 HelloUnity。



3）在该文件中输入以下代码：

```
using UnityEngine;

public class HelloUnity : MonoBehaviour
{
    private int counter;

    // Use this for initialization
    void Start()
    {
        print("Hello, Unity");
    }

    // Update is called once per frame
    void Update()
    {
        counter++;
        if (counter >= 60) {
            print("Count 60 times, reset");
      counter = 0;
        }
    }
}
```

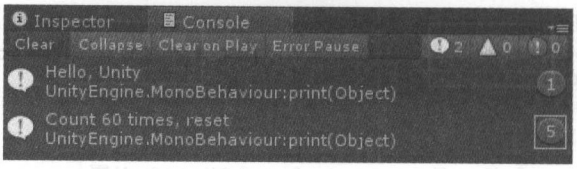
4）将代码文件拖至场景中的 Main Camera 上。



5）在 Inspector 面板上确定 Main Camera 上出现了 HelloUnity 组件。

6）点击运行按钮。

7）可以发现在 console 面板中开始打印出了“Hello, Unity”，接下来大概每一秒会打印一次“Count 60 times, reset”。



首先我们在 HelloUnity 类中继承了 MonoBehavior，添加了 Start 和 Update 方法。通过运行结果来解释一下 MonoBehavior 的事件机制。

- Start：在 GameObject 使能时运行，并且只运行一次，优先于所有的 Update 方法，因此可以将初始化的工作放入其中。
- Update：每帧画面渲染前调用，通常游戏需要稳定运行在每秒 60 帧左右，因此这里大概是每秒钟打印出一次。在 Update 方法中可以做很多检查和时刻更新的处理。

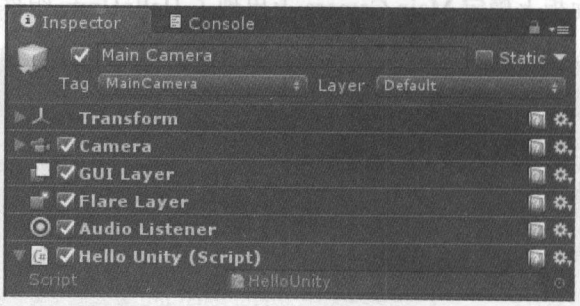
后文将详细介绍 MonoBehavior 的事件机制和如何使用 C# 脚本与 GameObject 做更多的交互。

1.3 Unity 3D 编程进阶

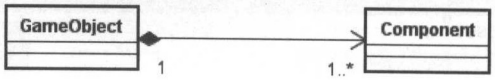
1.3.1 Unity 3D 的设计模式

Unity3D 的设计模式是一个典型的 Component 模式（设计模式是在软件设计中为了解决一些普遍存在的问题（大部分是为了软件的可维护性和健壮性）所产生的一门学科）。在 Unity 中，所有的对象都是 GameObject，每个 GameObject 都拥有一个以上 Component 的子类（Transform 是肯定有的）。

打开场景 1-2 Basic Script，选中 Main Camera，观察 Inspector。



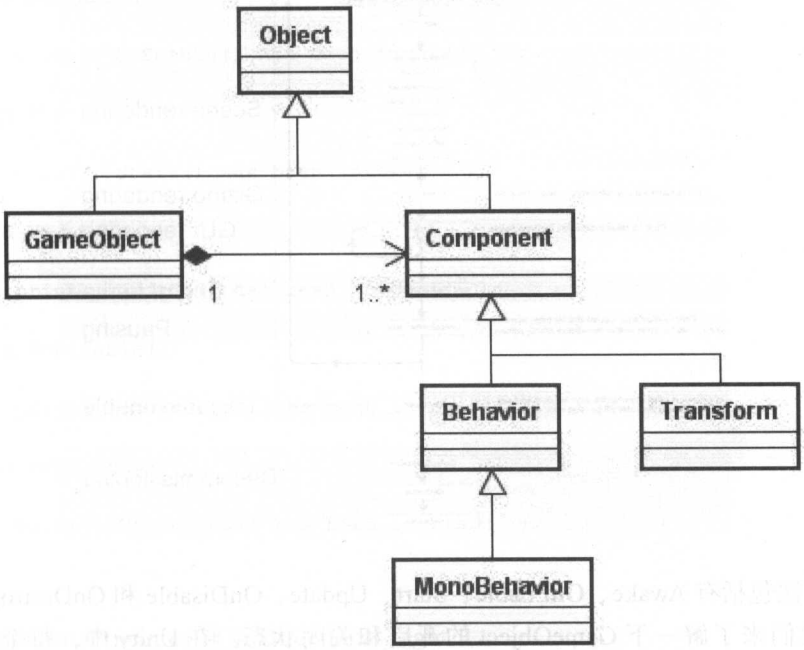
可以发现 Main Camera 这个 GameObject 拥有 Transform、Camera、GUI Layer、Flare Layer、Audio Listener 和 Hello Unity 这几个 Component 的子类。在 Unity 中所有的 GameObject 都是这样的结构：



这是GameObject的UML图，这里的Component代表Component的子类，如图所示GameObject和Component是一对多的关系。Main Camera这个GameObject在Inspector显示的内容也说明了这一点。

可以发现Hello Unity也被Unity当作了GameObject的一个Component，这是因为MonoBehavior也是Component的一个子类，因此可以拖到GameObject上添加为一个Component。通过点击Component右侧书的图标可以打开Unity的API说明文档，例如点击下图中的按钮可以打开MonoBehavior的说明文档：

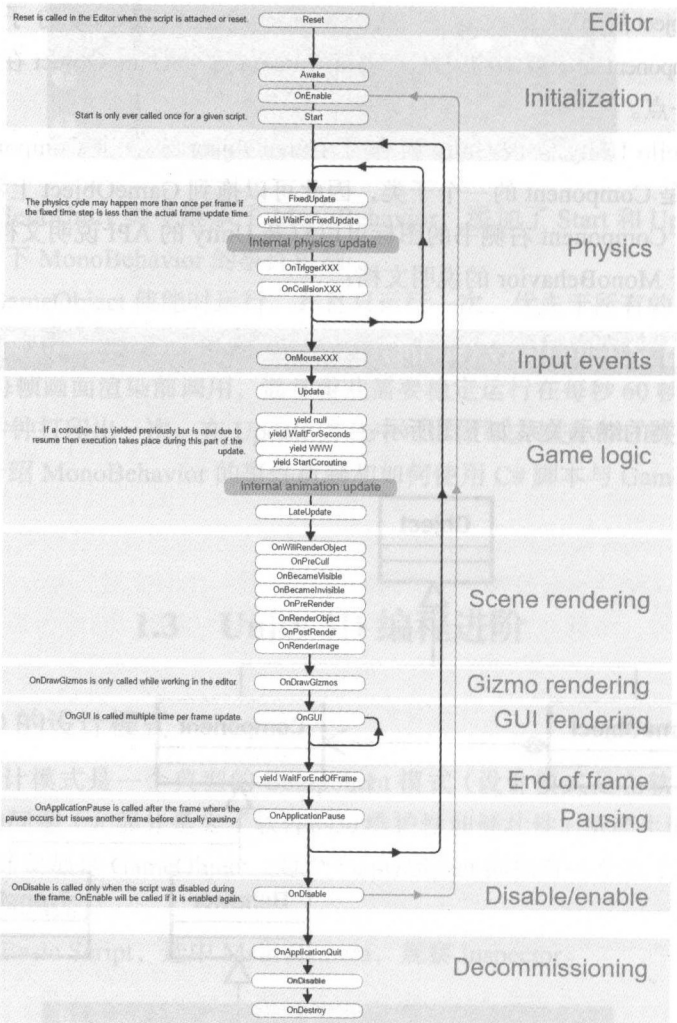
Unity中主要类的继承关系如下图所示：



图中的箭头表示继承关系，例如MonoBehavior是Behavior的子类，Behavior是Component的子类，Component是Object的子类。在Unity提供的类中，所有的类都是直接或间接地继承于Object。注意：是UnityEngine.Object并不是System.Object。与C#的System是不同的命名空间，由于Unity的类也是C#类的一部分，因此UnityEngine.Object肯定也继承于System.Object。

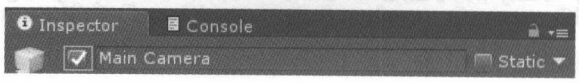
1.3.2 MonoBehavior的生命周期

MonoBehavior对象的生命周期（从生成到销毁）如下图所示：



主要方法包括有 Awake、OnEnable、Start、Update、OnDisable 和 OnDestroy。

首先我们来了解一下 GameObject 的开启和关闭状态。在 Unity 中，每个 GameObject 都有这两种状态。简单地说就像是一个开关：让引擎知道这个 GameObject 是否需要工作。实质上就是该 GameObject 的所有 Component 的 Update 函数是否需要被调用。例如对于 Camera 来说，是否需要继续渲染图像，对于 Renderer 来说是否继续显示在屏幕上。在 Inspector 中可以很方便地操作：选中或解除选中下图中的对勾即可。



在初始化中，主要涉及 Awake、OnEnable 和 Start 方法。三个函数在初始化时按顺序执行，但是只有 OnEnable 是有可能被重复执行的。因为 OnEnable 方法在 GameObject 关闭并再次开启时会被调用，正如它的名字，在每次开启时都会被调用，OnDisable 同理：每次被

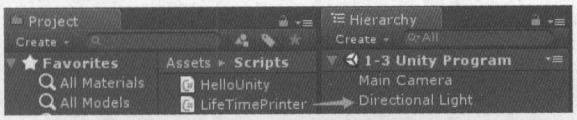
关闭时都会被调用。Update 每帧渲染前都会被调用，但如果 GameObject 是关闭状态的话，Update 并不会被调用。

我们可以通过以下程序来说明 MonoBehavior 的生命周期和对应事件。

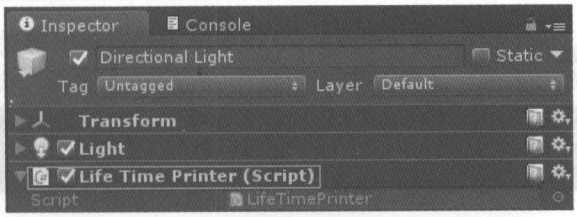
1）在 Assets/Scenes 目录下新建一个场景 1-3 Unity Program，在 Assets/Scripts 文件夹下创建脚本 LifeTimePrinter.cs，添加以下代码：

```
using UnityEngine;

public class LifeTimePrinter : MonoBehaviour
{
    void Awake()
    {
        print("Awake() has been invoked!");
    }
    void OnEnable()
    {
        print("OnEnable() has been invoked!");
    }
    void Start()
    {
        print("Start() has been invoked!");
    }
    void Update()
    {
        print("Update() has been invoked!");
    }
    void OnDisable()
    {
        print("OnDisable() has been invoked!");
    }
    void OnDestroy()
    {
        print("OnDestroy() has been invoked!");
    }
}
```

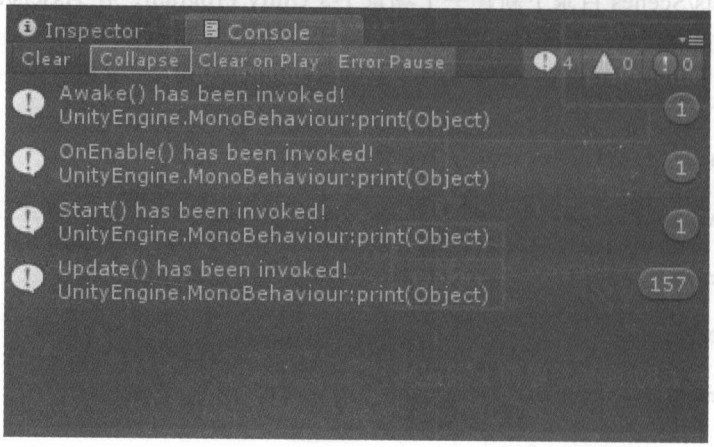
2）将 LifeTimePrinter.cs 拖至 Directional Light 上。



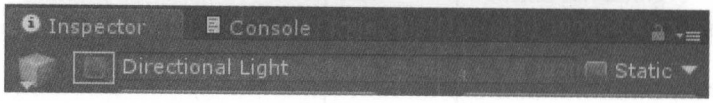
3）点击 Directional Light，在 Inspector 面板中确定脚本已经添加。



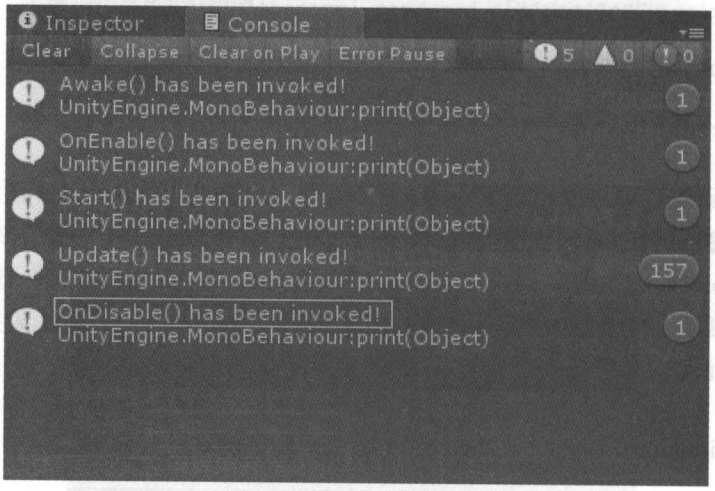
4）点击运行执行程序，观察 Console 面板显示的内容（注意这里的 Collapse 选项是打开状态，因此 Log 都是以数量的形式显示右侧，否则 Update 的消息会全部展开刷屏不利于观察），发现 Awake、OnEnable 和 Start 依次被调用，Update 在持续地被调用。



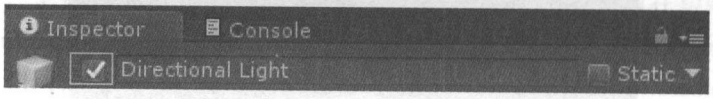
5）关闭 Directional Light。



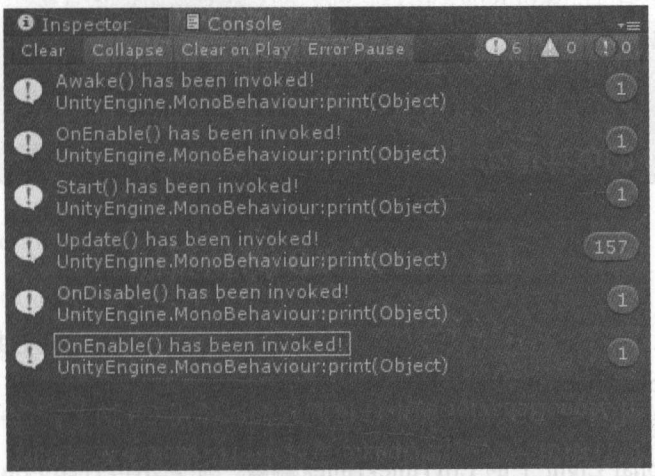
6）观察发现 OnDisable 被调用了。



7）打开 Directional Light。

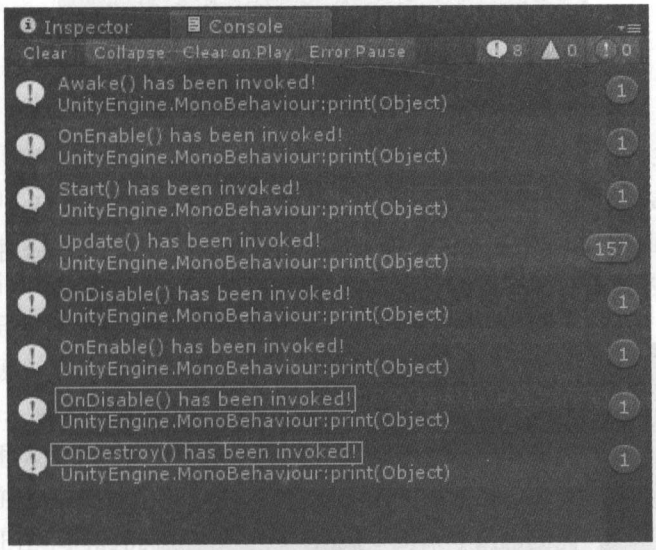


8）观察发现 OnEnable 被调用了。



9）删除 Directional Light（在 Hierarchy 中选中点击 Delete 键或单击鼠标右键并点击 Delete）。

10）观察发现 OnDisable 和 OnDestroy 被调用了。



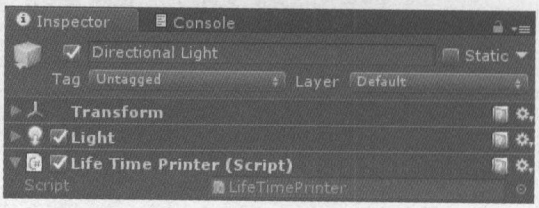
由此例可以看出从创建到销毁的过程中，MonoBehavior 核心事件的调用时机和次序。

1.3.3 控制 GameObject 的位置

在 Unity 中，只要调用负责特定功能的 Component 的子类其中的方法或变量，就可以控制 GameObject 的行为。例如 Transform 中提供了变换位置、角度、大小的方法及相关的数据，Trigger 提供了物体碰撞的开启或关闭，以及碰撞器的形状和大小。

本节主要介绍如何使用 Transform 控制物体的移动，在 MonoBehavior 中有一个成员

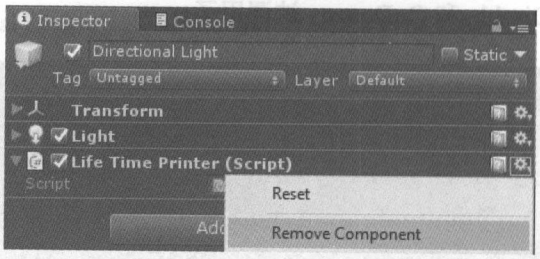
变量为 transform，其实就是该 MonoBehavior 所属的 GameObject 的 Transform 对象。因为 GameObject 是肯定有 Transform 的，所以 Unity 为了方便调用把它直接作为了 MonoBehavior 的成员变量。例如在上面的例子中：



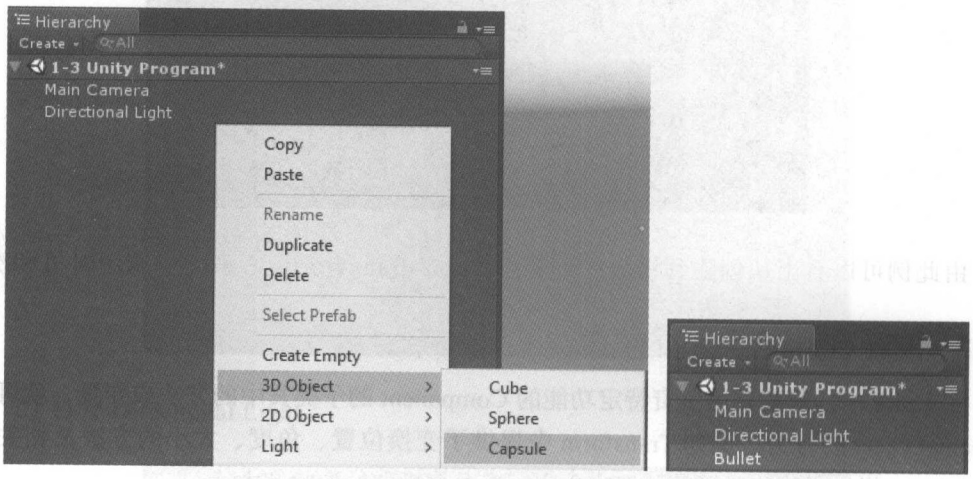
在 Directional Light 这个 GameObject 上的 LifeTimePrinter 可以直接使用成员变量 transform 来控制该 GameObject 的位置、旋转和大小。

下面我们通过一个程序来说明它是如何工作的。

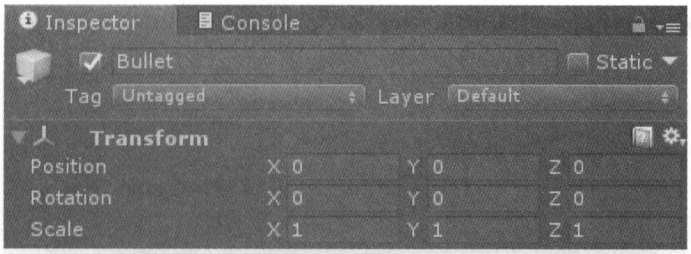
1）将 Directional Light 上的 LiftTimePrinter 删除，使它不再打印 Log。点击右侧的小齿轮，选择 Remove Component。



2）在场景中创建一个 Capsule：在 Hierarchy 的空白处点击鼠标右键，依次选择 3D Object → Capsule，并将其重命名为 Bullet（子弹）。



3）选中 Bullet，在 Inspector 面板中将 Transform 设置为下图所示的值：

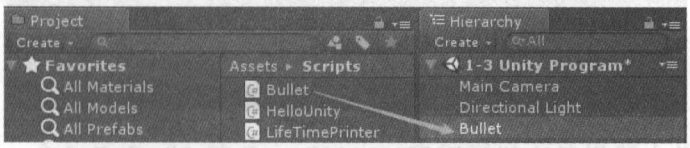


4）在 Assets/Scripts 文件夹中创建 C# 脚本 Bullet.cs，输入以下代码：

```
using UnityEngine;

namespace Assets.Scripts
{
    public class Bullet : MonoBehaviour
    {
        public Vector3 speed = Vector3.up;

        void Update()
        {
            Vector3 moveDistance = speed * Time.deltaTime;
            transform.Translate(moveDistance);
        }
    }
}
```

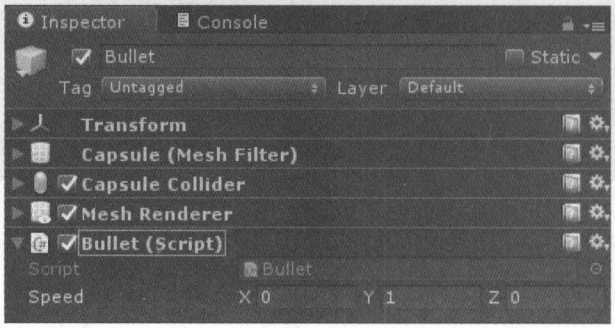
可以看到在代码中我们给了 Bullet 一个成员变量 speed，类型为 Vector3。即一个三维的向量，包括 (x, y, z)。这个变量控制 Bullet(子弹）的移动速度，默认值为 Vector3.up 即向上。在 Update 中 transform.Translate 函数可以移动 GameObject 输入 Vector3 的距离，但是为什么移动距离乘以了一个 Time.deltaTime 呢？这是因为需要适配不同运行速度的设备：因为不同机器的性能不同，导致不同机器在 1 秒内可以渲染的帧数是不同的，因此 Update 的调用数量也不同。举例来说一台配置很高的机器 A 一秒可以渲染 100 帧，一台配置很低的机器 B 一秒只能渲染 20 帧，那么如果这里的代码直接写 speed，A 机器中这个子弹会移动 100 个单位，但 B 机器只能移动 20 个单位。这明显是不合理的。Time.deltaTime 就是距离上一帧渲染到现在的时间，在这里用速度乘以一帧之间的时间间隔才应该是这一帧之内所应该前进的距离，这样也能保证在不同的机器上的这个子弹的运动速度是相同的。

5）将 Bullet.cs 文件拖至 Bullet 上。

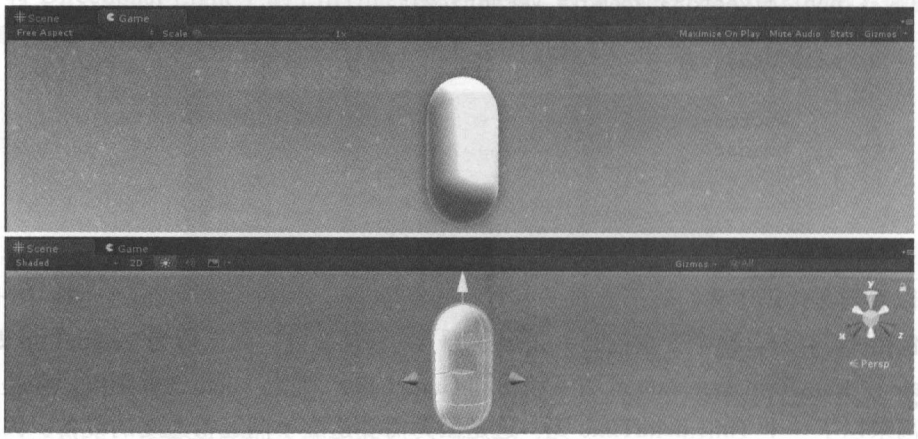


6）确认 Bullet 上有 Bullet 脚本，可以发现 Bullet 中的 speed 变量显示到了 Inspector 中，这是因为 Unity 的序列化机制可以让 Component 的子类中的公有变量显示在 Inspector 中并

可以在编辑器中修改。这是一个非常便利的机制，在开发过程中经常使用。



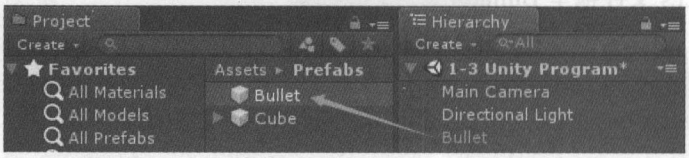
7）点击运行，可以观察到 Bullet 会一直向上运动一直到屏幕外，在 Scene 面板中可以继续观察子弹向上移动的过程。



1.3.4 控制 GameObject 的生成和销毁

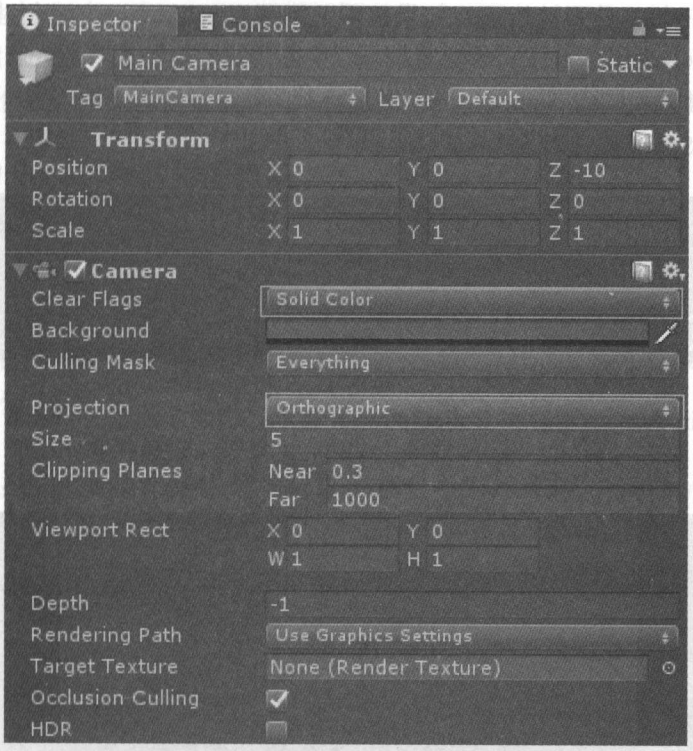
在游戏编程中往往需要控制 GameObject 的生成和销毁，例如这个子弹应该是从某一个地方生成，当超过了视野范围则销毁。通过下面的例子来了解在 C# 脚本中应该如何实现。

1）将上一节中的 Bullet 拖至 Prefabs 文件夹并生成对应的 Prefab。



2）删除 Hierarchy 中的 Bullet。

3）调整 Camera 的参数：将 Clear Flags 设为 Solid Color，将 Projection 设为 Orthographic。正投影相机的尺寸没有透视因此便于计算。将 Transform 的 Y 坐标设为 0：



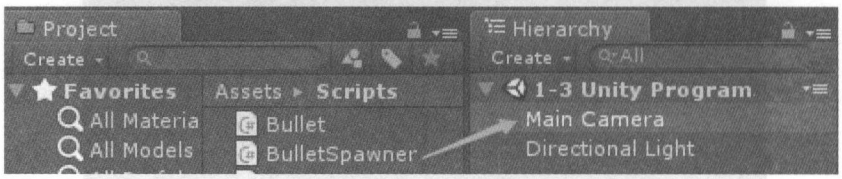
4）在 Assets/Scripts 文件夹下创建 BulletSpawner.cs 文件，为其添加以下代码：

```
using UnityEngine;

public class BulletSpawner : MonoBehaviour
{
    public GameObject bulletPrefab;

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space)) {
            CreateBullet();
        }
    }

    void CreateBullet()
    {
        GameObject bullet = Instantiate(bulletPrefab);
        bullet.transform.position = Vector3.zero;
    }
}
```

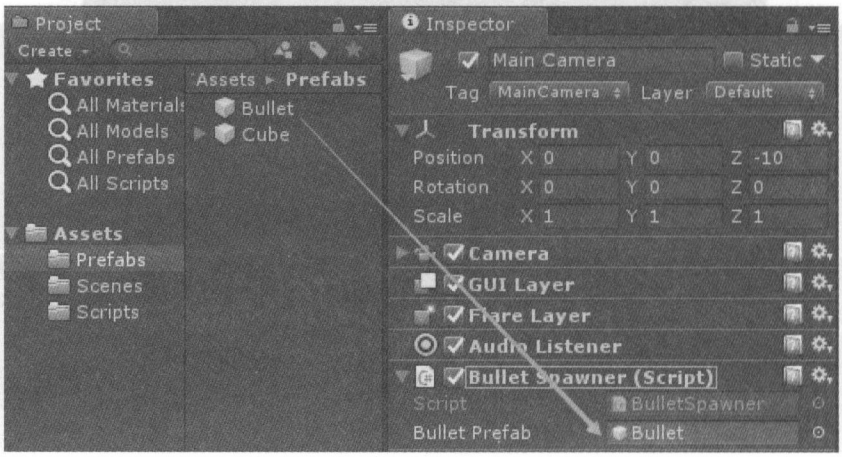
在该类中有一个 bulletPrefab 的 GameObject，我们可以在编辑器中将 Bullet 的 Prefab 拖到 Inspector 的对应位置，来告诉代码这里的 GameObject 到底是谁。在 Update 中，Input.GetKeyDown 方法返回一个布尔值，即 true（真）或 false（假）来检测输入参数的按

键（本例中是空格键）是否被按下，如果按下，调用 CreateBullet 方法创建一个 bullet。在 CreateBullet 方法中调用了 Instantiate 方法，该方法可以根据输入参数（通常是 prefab 的引用）创建一个与参数相同的 GameObject。随后将其的位置设置为（0，0，0）。

5）将 BulletSpawner.cs 拖至 Main Camera 上。



6）确定 Main Camera 上存在 BulletSpawner 脚本，将刚才创建的 prefab 拖至 bulletPrefab 变量上。



7）将 Bullet.cs 中的 Update 函数改为以下代码：

```
void Update()
{
    Vector3 moveDistance = speed * Time.deltaTime;
    transform.Translate(moveDistance);

    if (transform.position.y >= Camera.main.orthographicSize) {
        Destroy(gameObject);
    }
}
```

这里的处理就是在 Update 后增加了一个判断，当前位置的 y 坐标是否已经超过了 Camera 的可视范围，如果已经超过则销毁这个对象。可以使用代码中的方式来获取当前的主相机对象。

8）点击运行，按空格键会发现屏幕中间生成了一个 Bullet，移动到屏幕顶端时消失，

在 Hierarchy 中也消失，证明 Bullet 已经被销毁。

可以注意到 Bullet 在完全从视野中消失之前就被销毁了，因为 Transform 的位置位于胶囊体的中心，我们的判断条件是胶囊中心的位置大于相机尺寸即销毁 Bullet。

1.3.5 处理 Unity 3D 中的物体碰撞

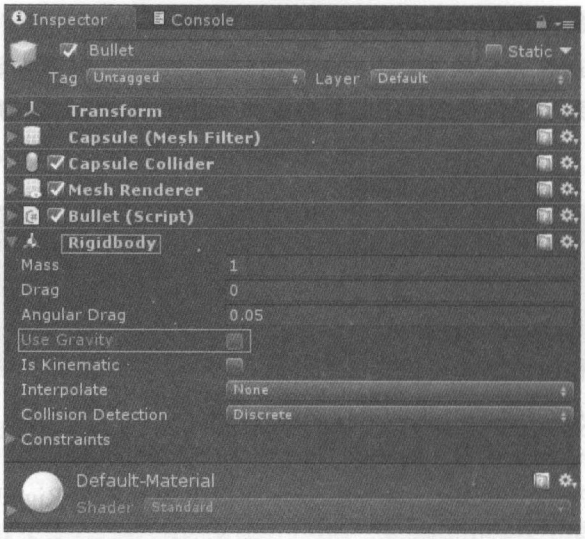
碰撞检测是游戏引擎中物理引擎的主要工作之一，也是非常常用的功能之一。现在市面上有很多成熟的物理引擎可以实现高效的碰撞检测并模拟真实世界的反馈。Unity 使用的物理引擎 NVIDIA 的 PhysX。

在 Unity 中碰撞检测的基本单位是 Rigidbody，仅拥有 Rigidbody 的 GameObject 才会检测其碰撞，并通知 MonoBehavior 中的事件函数。

我们可以通过一个例子简单了解一下 Unity 中碰撞检测的机制。

1）选中 Assets/Prefabs/Bullet，在 Inspector 面板中为其添加 Rigidbody 组件，依次点击 Add Component → Physics → Rigidbody。

2）添加成功后组件会出现在 Inspector 面板中，将 Use Gravity 选项去掉。保留的话该 GameObject 会受到重力不断下降，这会干扰运动部分的代码。



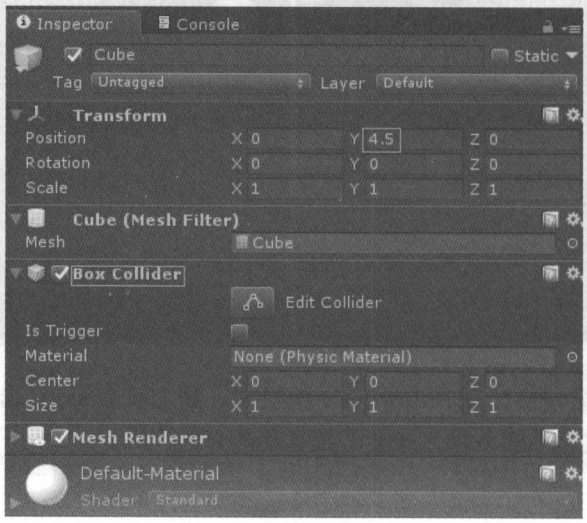
3）打开 Bullet 脚本，添加以下函数：

```
void OnCollisionEnter(Collision collision)
{
    Destroy(gameObject);
}
```

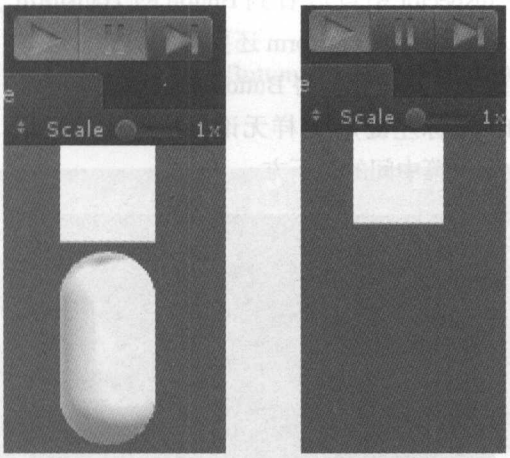
OnCollisionEnter 会被 Unity 在检测到的时候自动调用，在其中我们直接把这个对象销毁，也就是说，只要碰到任何碰撞体（Collider）我们就直接把对象销毁。

4）在场景中创建一个 Cube，在 Hierarchy 空白处点击鼠标右键，并依次选择 3D Object → Cube。

5）选中 Cube，调整位置至视野最上方。可以发现 Cube 是含有一个 Box Collider 的组件，因此如果 Bullet 与其发生碰撞是可以被检测到的，Bullet 会被销毁。



6）点击运行，点击空格生成 Bullet，可以发现在碰到这个 Box 的一瞬间 Bullet 将被销毁了。



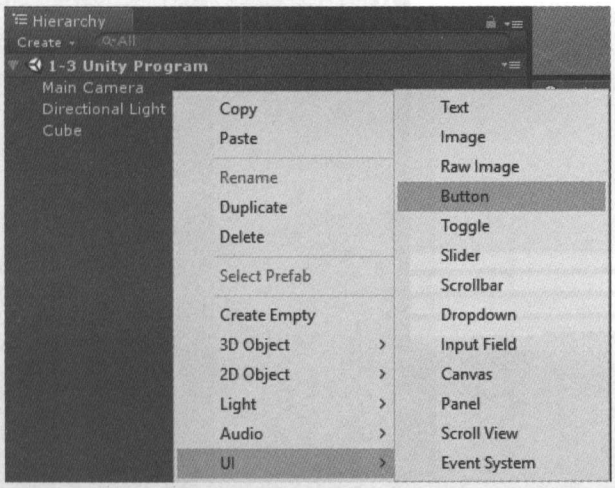
类似的函数还有 OnCollisionExit、OnCollisionStay、OnTriggerEnter、OnTriggerStay 和 OnTriggerExit。具体的使用方法可以参考 Unity 的文档。

1.3.6 UI 组件的使用

UI 是游戏开发必要的一环，Unity 从 4.6 开始提供的新一代 uGUI 系统功能十分强大而且设计精巧。程序结构比较类似于之前的一个 UI 插件 NGUI。

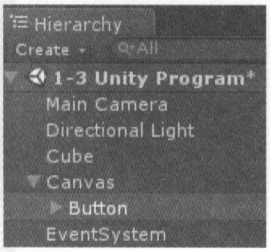
本节主要介绍一下 Button 这个最常用的 UI 组件的使用方法。

1）首先创建一个 Button（按钮），在 Hierarchy 面板的空白区域上点击鼠标右键，依次选择 UI → Button。

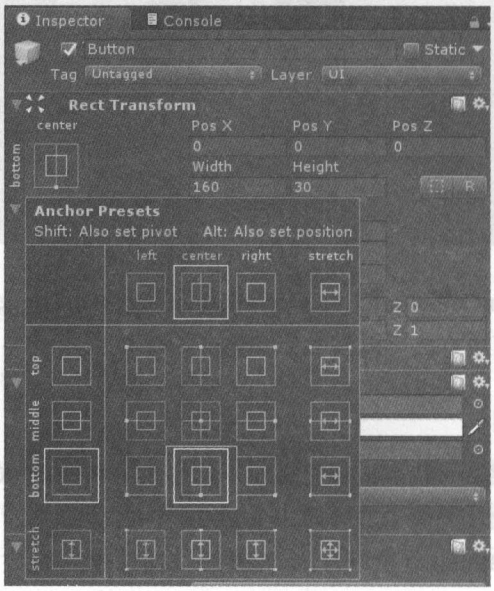


2）Unity 会自动创建一个 Canvas（画布）并将 Button 放在 Canvas 下面。这里的画布

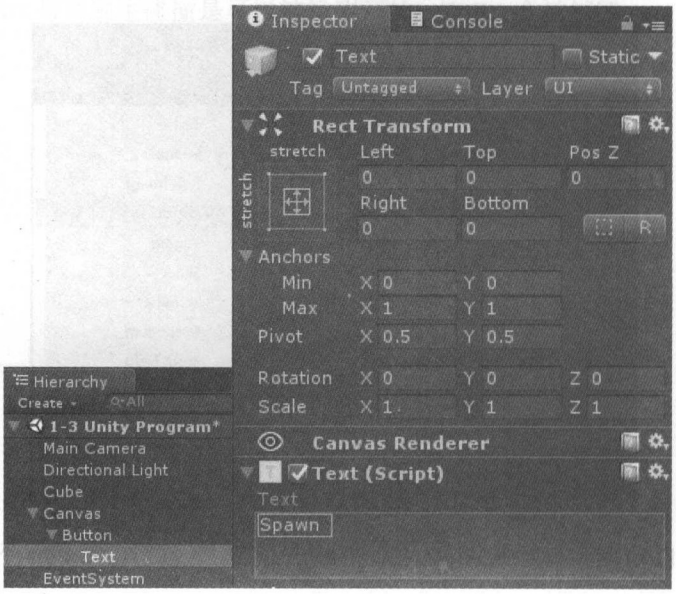
就是 UI 元素的集合，UI 元素必须在 Canvas 下面。还创建了一个 EventSystem 负责 UI 模块中处理事件的通信。



3）选中 Button，在 Inspector 中可以看到 Button 的 Transform 被换成了 RectTransform，而且有比 Transform 还要多的属性，这些属性都是用于描述 UI 位置的，这里我们将 Button 的锚点置于屏幕下方（同时按住 Alt+Shift+ 鼠标左键），这样无论屏幕尺寸如何变化，这个按钮都始终会位于屏幕中间的最下方。



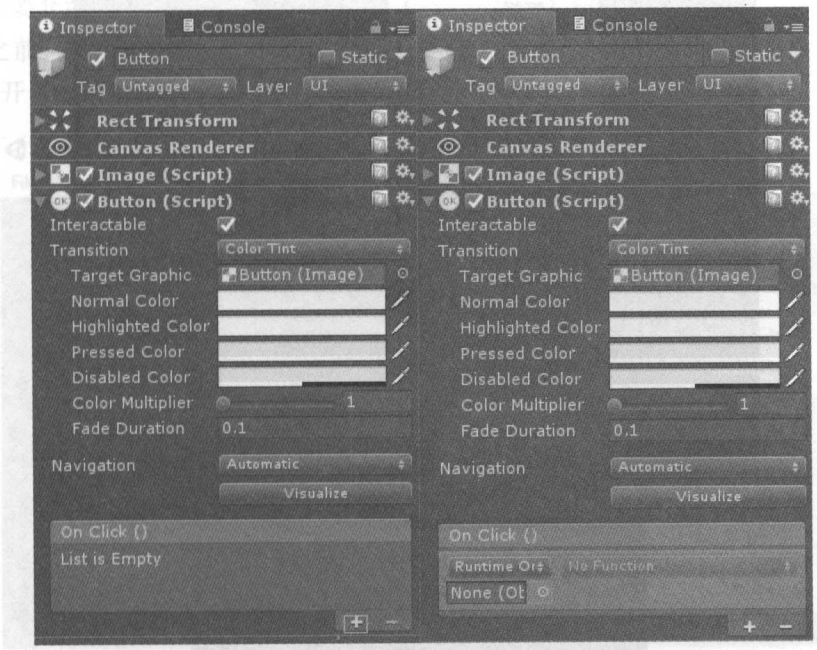
4）选中 Button 下面的 Text，将内容改为 Spawn。



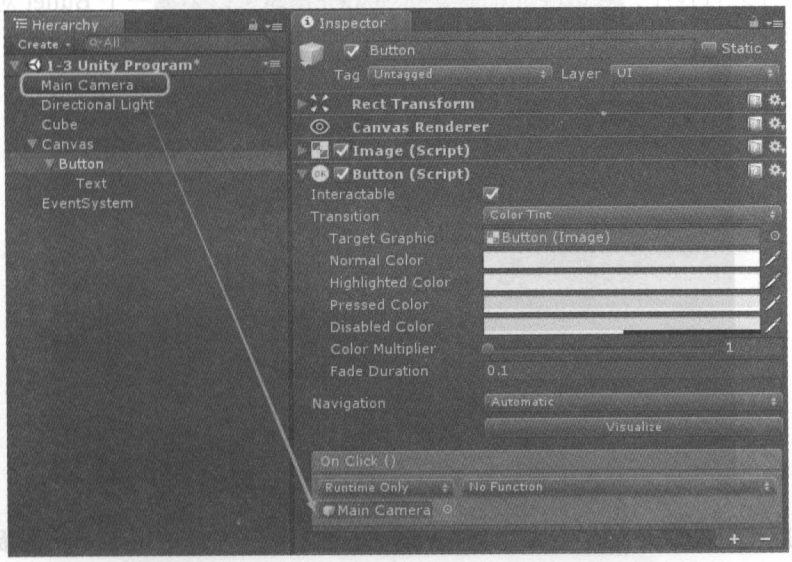
5）将 BulletSpawner.cs 的成员方法 CreateBullet 设为公有的（因为 Button 的 Click 事件响应函数需要时公有对外可见的）。

```
public void CreateBullet()
```

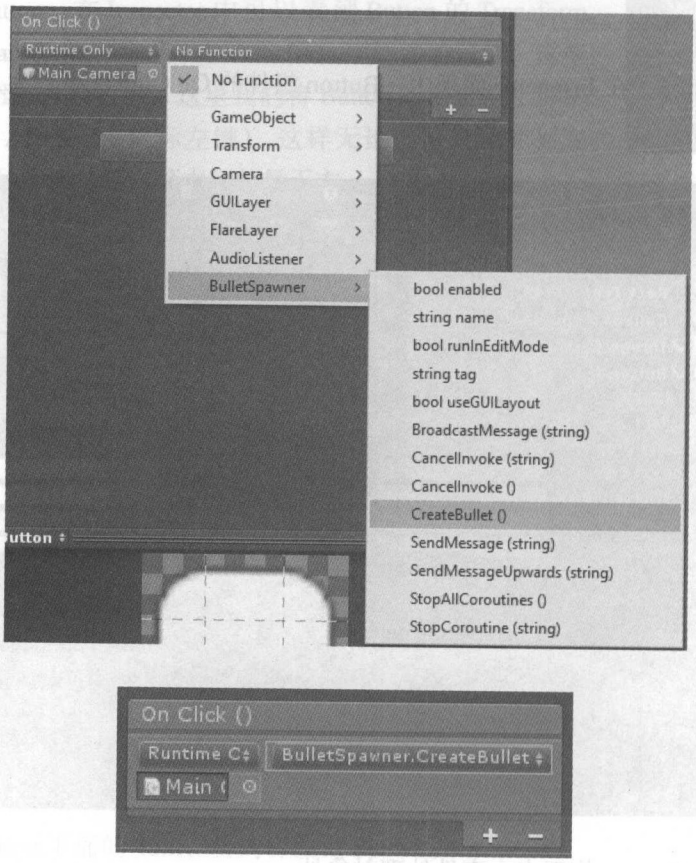
6）选中 Button，在 Inspector 面板中，Button 组件的 On Click() 事件上点击“+”，添加点击事件响应方法。



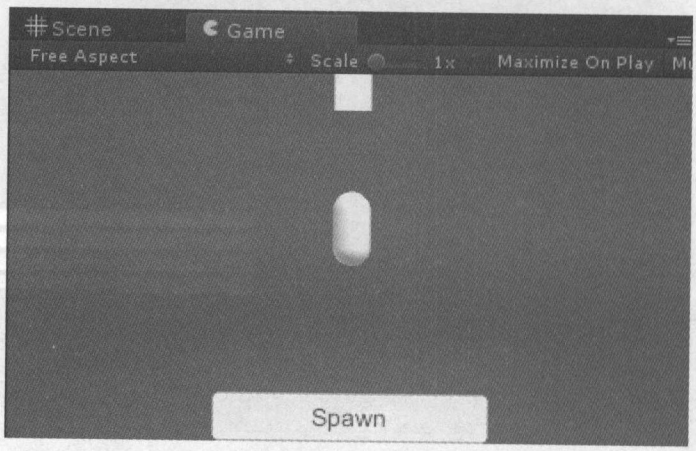
7）将 Main Camera 拖至点击事件处理对象处。



8）选择 BulletSpawner 组件的 CreateBullet 方法为点击事件响应方法，通过该机制可以在该按钮被点击时调用选择的方法。



9）保存并运行程序，点击屏幕正下方的 Spawn 按钮可以生成一个 Bullet 对象。



Unity 的 UI 组件就是利用这种事件通知机制进行交互的：在 Inspector 面板中绑定响应

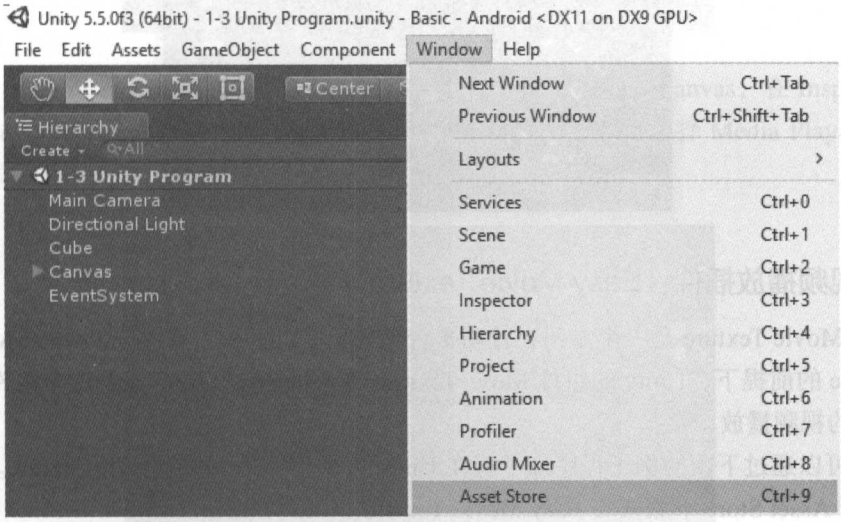
GameObject，在相应方法中选择特定组件的方法。其他的 UI 组件例如 Toggle、Slider 等也是使用类似的方式交互。具体方式可以查看 Unity 的 API 文档。

1.4 AR 中常用的 Unity 3D 插件

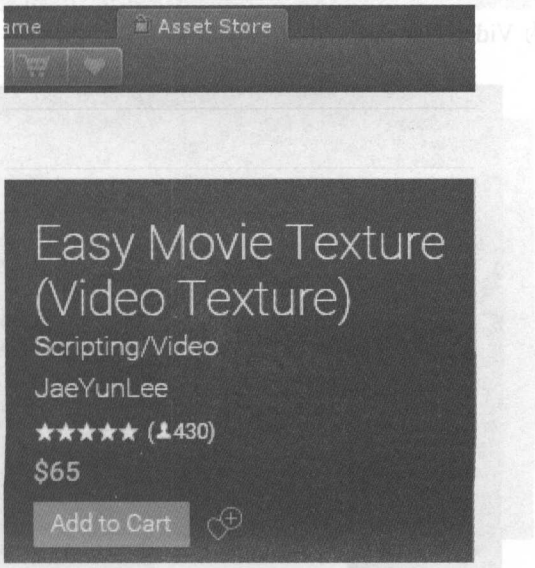
在 Unity 开发过程中往往需要使用到一些第三方插件，可以使开发的过程更加简单和高效，本章主要介绍几个十分常用的插件。

在此之前，先简单介绍一下插件的下载与安装。

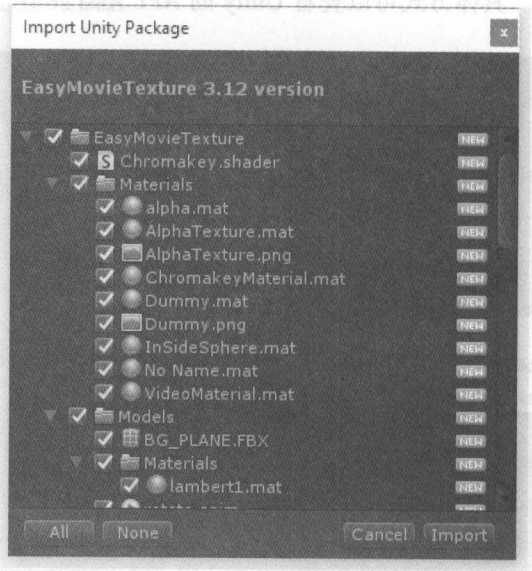
1）打开 Asset Store，在菜单栏中点击 Window → Asset Store。



2）找到需要的插件，购买后下载（免费插件可以直接下载）。



3）下载完成后会弹出以下窗口，点击 Import。



1.4.1 视频播放插件：Easy Movie Texture

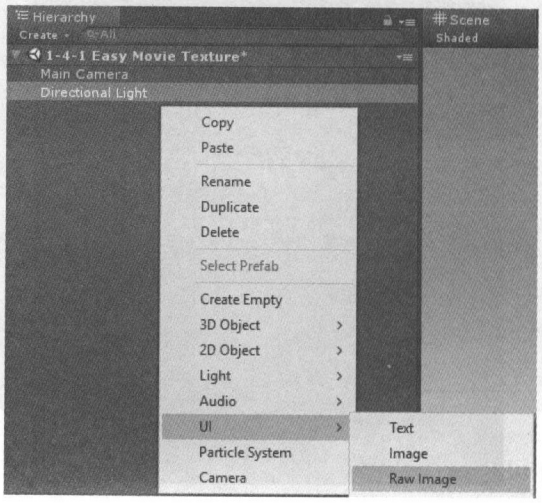
Easy Movie Texture 是一个专门在移动平台播放视频的插件。在 Windows 或 Mac 上装有 Quick Time 的前提下，Unity 的组件 MovieTexture 将可以播放视频。但是这个组件并不支持移动平台的视频播放。

我们可以通过下面的例子程序来了解在 Unity 中如何使用 Easy Movie Texture。

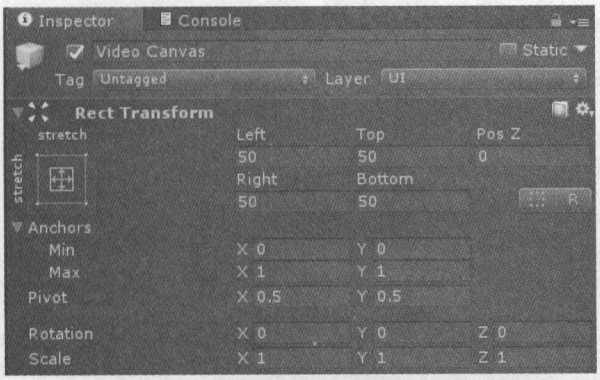
1）在 Asset Store 下载安装 Easy Movie Texture。

2）在 Assets/Scenes 目录下创建一个新场景，名为 1-4-1 Easy Movie Texture。

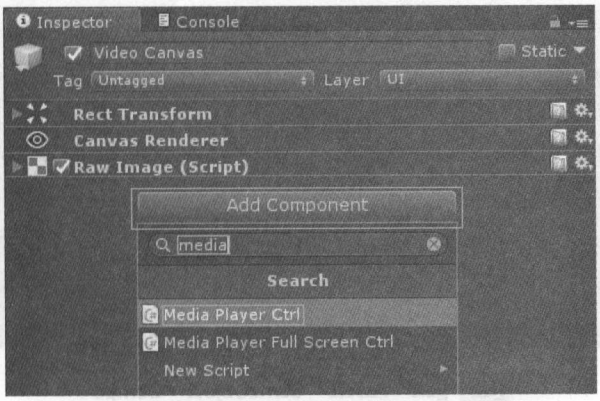
3）创建 UI 组件 RawImage。在 Hierarchy 面板空白处点击右键，依次选择 UI → Raw Image，并将其命名为 Video Canvas。



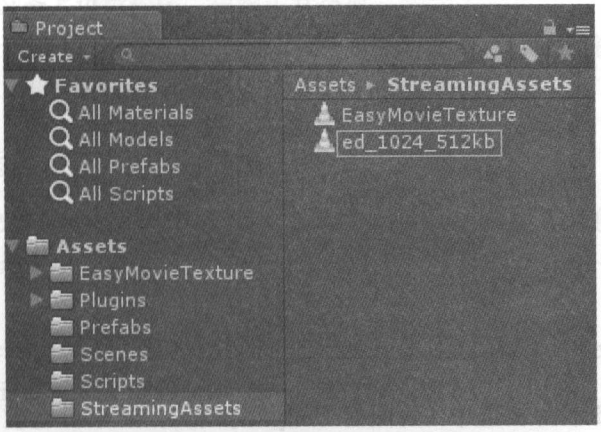
4）选中 Video Canvas，调整 Rect Transform，如下图所示：

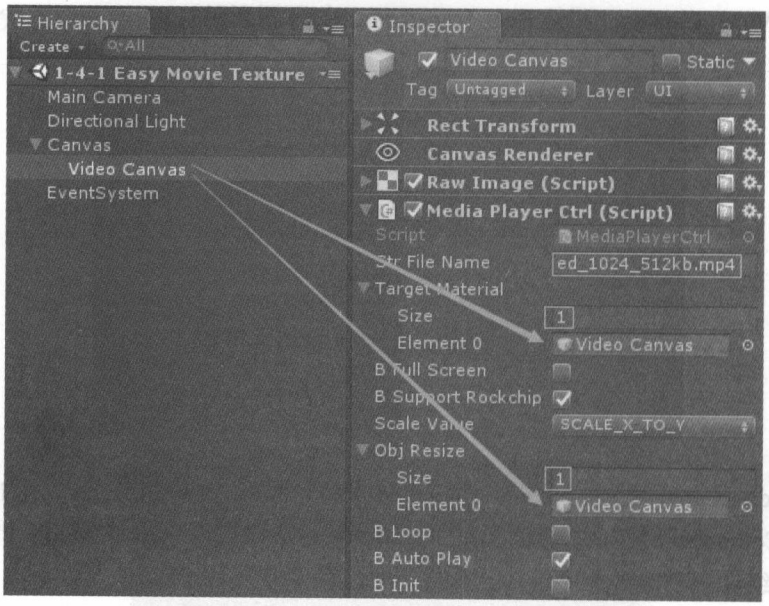


5）为 Video Canvas 添加 Media Player Ctrl 组件：选中 Video Canvas，在 Inspector 面板中点击 Add Component 按钮，在输入框中输入 media，在候选里选择 Media Player Ctrl（这是 Unity 添加组件一个比较快捷的方式）。



6）设置 Media Player Ctrl 组件：

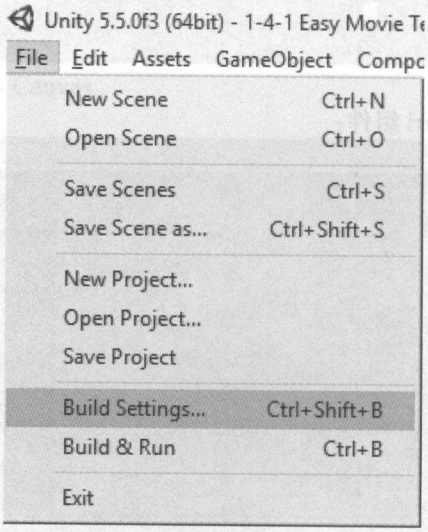




在 Str File Name 中填入 ed_1024_512kb.mp4，这是位于 Assets/StreamingAssets 目录下的视频文件名，本例中是 Easy Movie Texture 中自带的视频文件，建议选用通用的视频编码（H264）保证大部分设备可以正常播放指定的视频文件。将 Target Material 和 Obj Resize 的 Size 都改为 1，然后将 Video Canvas 对象拖至两处的 Element 0 位置。这是最终播放视频的渲染对象，可以是多个对象。

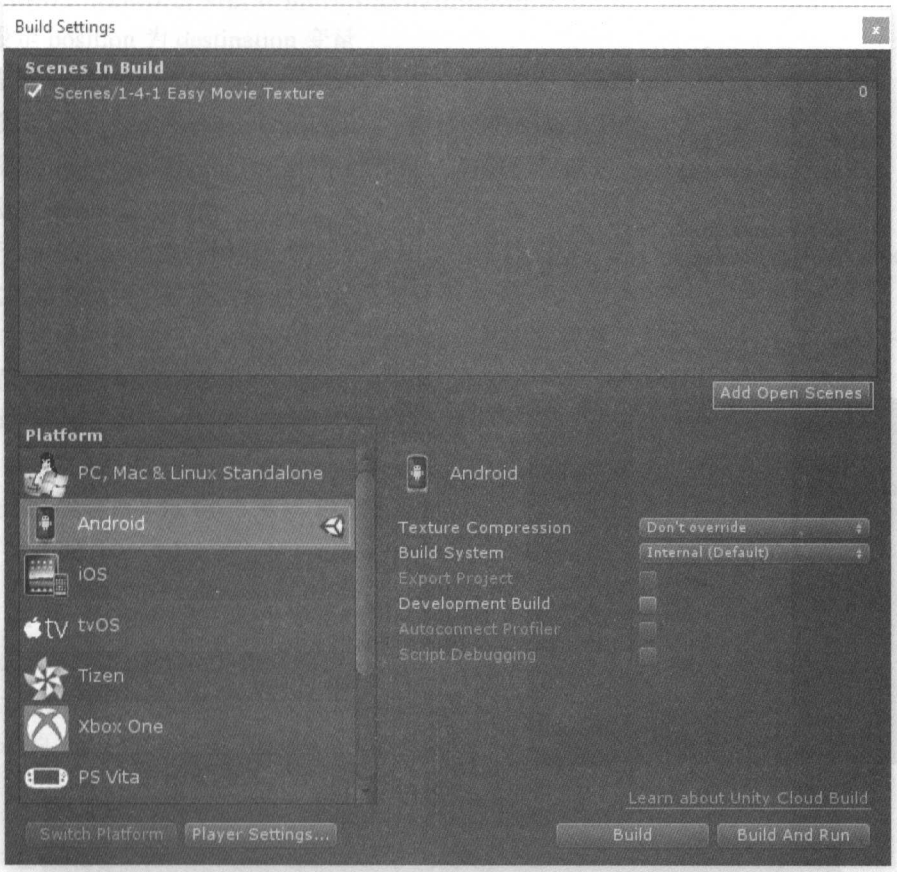
7）配置 Unity 的安卓开发环境，网上有很多资料，这里不再赘述。

8）在菜单栏依次点击 File → Build Settings 打开编译窗口。



9）确定 Android 平台被选中（后面有 Unity 的图标），如果没有被选中，选择 Android

平台并点击左下角的 Switch Platform。等待编译完成后，点击 Add Open Scenes 将打开的场景加入编辑场景列表。



10）点击 Build 生成 APK，或者在连接 Android 设备且打开 USB 调试的状态下点击 Build And Run 直接部署至 Android 设备并运行。

11）在 Android 设备上运行程序，视频正常播放。

可以注意到在一开始 Easy Movie Texture 自动把上下缩小了以适应视频的比例，这个行为是由 Media Player Ctrl 的 Scale Value 变量决定的。

1.4.2 动画控制插件：iTween

iTween 是一个免费且开源的处理简单动画的类，使用起来也非常便利。Tween 这个词就是补间动画，意思是生成关键帧之间需要的过渡数据。本节重点介绍一下这个类的使用方法。

1）在 Assets/Scenes 目录下创建一个新的场景，并命名为 1-4-2 iTween。

2）在 Assets/Scripts 目录下创建脚本 TweenTest.cs，输入以下代码：

```
using UnityEngine;
using System.Collections;

public class TweenTest : MonoBehaviour
{
    public Vector3 destination;
    public Vector3 rotation;

    Hashtable moveArgs;
    Hashtable rotateArgs;

    Vector3 initPosition;
    Vector3 initRotation;


    void Start()
    {
        initPosition = transform.position;
        initRotation = transform.eulerAngles;

        // Setup move arguments
        moveArgs = new Hashtable();
        moveArgs.Add("position", destination);
        moveArgs.Add("speed", 1);
        moveArgs.Add("easeType", iTween.EaseType.easeInOutExpo);
        moveArgs.Add("loopType", iTween.LoopType.pingPong);
        moveArgs.Add("delay", 0.1);

        // Setup rotate arguments
        rotateArgs = new Hashtable();
        rotateArgs.Add("rotation", rotation);
        rotateArgs.Add("easeType", iTween.EaseType.easeInOutBack);
        rotateArgs.Add("loopType", iTween.LoopType.pingPong);
        rotateArgs.Add("delay", 0.4);
    }

    public void OnClickMove()
    {
        transform.position = initPosition;
        iTween.Stop(gameObject);
        iTween.MoveTo(gameObject, moveArgs);
    }

    public void OnClickRotate()
    {
        transform.eulerAngles = initRotation;
        iTween.Stop(gameObject);
        iTween.RotateTo(gameObject, rotateArgs);
    }
}
```

首先定义两个 Vector3 类型的位置变量 destination（目的地）和 rotation（旋转量），并将

它们设为共有变量使得它们可以被外部修改。随后在 Start 方法中首先记录 GameObject 的初始位置和角度，然后初始化移动和旋转的参数，参数类型是一个散列表的集合，可以简单地理解为键值对。例如在移动参数 moveArgs 的初始化中：

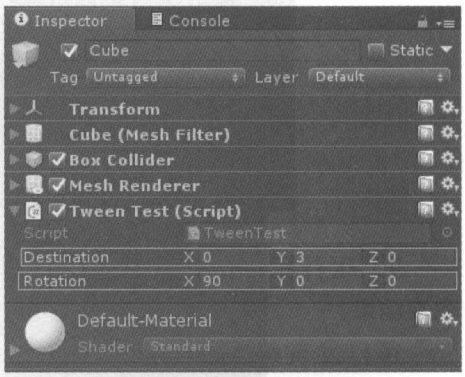
- 设定 position 为 destination 变量
- 设定 speed 为 1
- 设定 easeType 为 easeInOutBack（一种移动的趋势，具体参考文档）
- 设定 loopType 为 pingPong（来回循环，类似乒乓球）
- 设置 delay 为 0.1 秒

对于 rotateArgs 也进行类似的初始化。在点击事件响应方法 OnClickMove 和 OnClickRotate 中利用这些参数就可以控制对象的运动了。

3）在场景中创建一个 Cube：在 Hierarchy 中单击右键，依次选择 3D Object → Cube，将其放置于起点（0，0，0）。

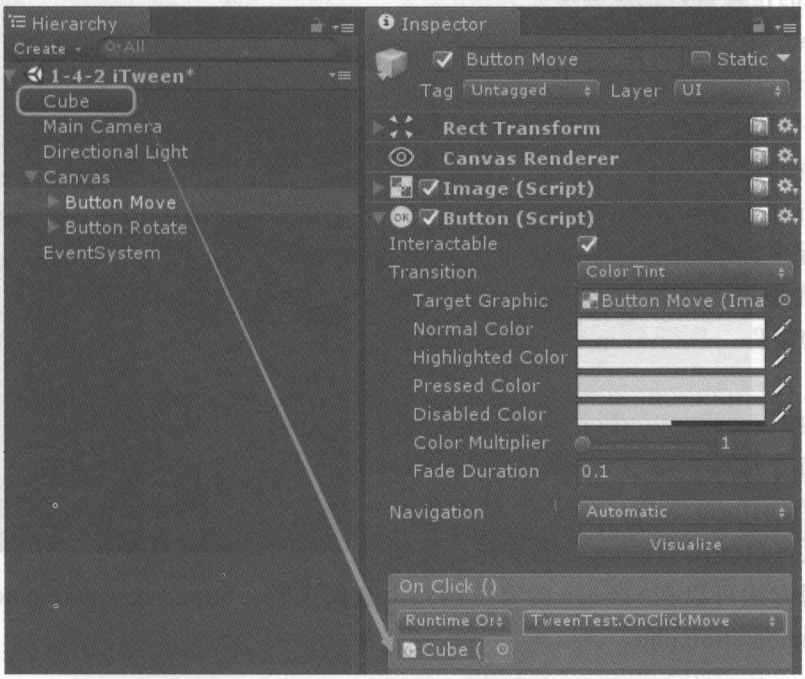
4）将 TweenTest.cs 拖至刚才创建的 Cube。

5）确定 Main Camera 上有 Tween Test 组件，设定 destination(终点）为（0,3,0），设定 rotation(旋转角度）为（90，0，0）。

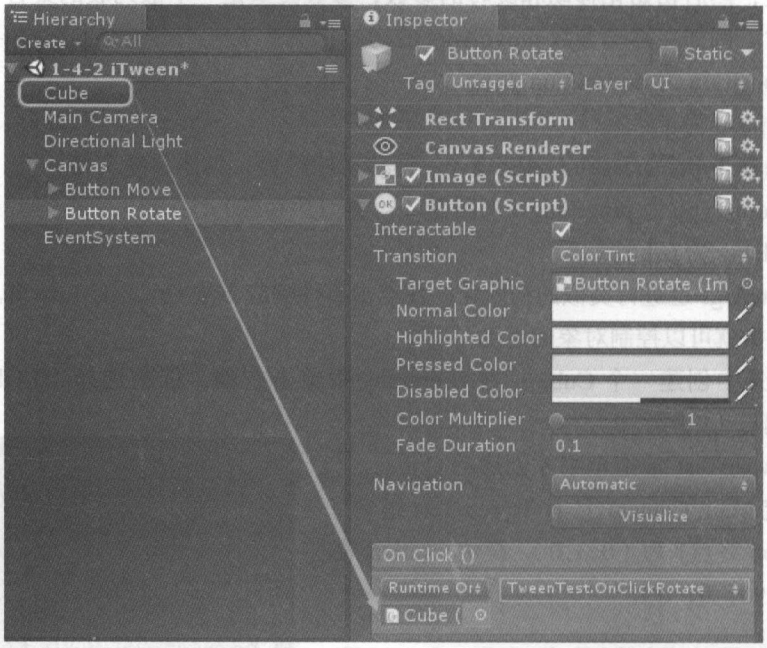


6）在场景中创建两个 Button，Move 和 Rotate，并更新对应 Test，调整其位置。

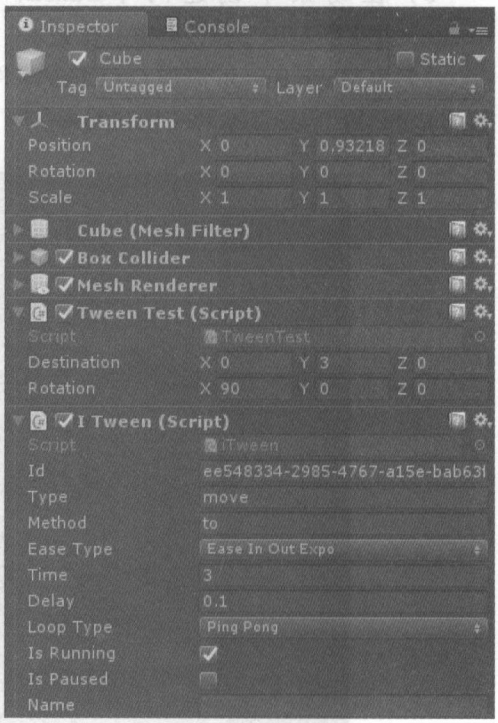
7）为 Move Button 添加 On Click 事件处理函数为 Cube 的 TweenTest.OnClickMove。



8）为 Rotate Button 添加 On Click 事件处理函数为 Cube 的 TweenTest.OnClickRotate。



9）运行程序，点击 Move 和 Rotate 按钮观察结果。



可以发现每次点击 Move 时 Cube 都会回到开始所在的位置进行往返运动，类似地，每次点击 Rotate 时 Cube 都会回到初始角度进行往返旋转。通过以上的方式为 iTween 设置移动的参数，执行特定的函数便可以使 GameObject 执行特定的动画。

在点击时观察 Cube 的 Inspector 可以发现：在运行中，Cube 上多了一个 i Tween 的组件，这就是调用 MoveTo 和 RotateTo 方法动态添加的组件，执行 Stop 方法会删除这个组件。

使用 iTween 可以让 GameObject 的一些常用属性随着时间以特定的规律变化，大部分应用于 Transform 组件，但是针对颜色和其他组件也可以使用 iTween 动态变化，更加详细的使用方法可以在 iTween 的官方网站上查阅文档：http://www.pixelplacement.com/itween/documentation.php。

1.4.3 手势控制插件：Easy Touch

在之前的项目中，我们使用了 Input 类来获取外界输入。但是当需要获取多点触控的数据和进行相关的判断时，使用 Unity 的内部类不是非常方便。因此 Easy Touch 应运而生，使用 Easy Touch 可以十分便捷地获取并判断多点触控相关的操作。

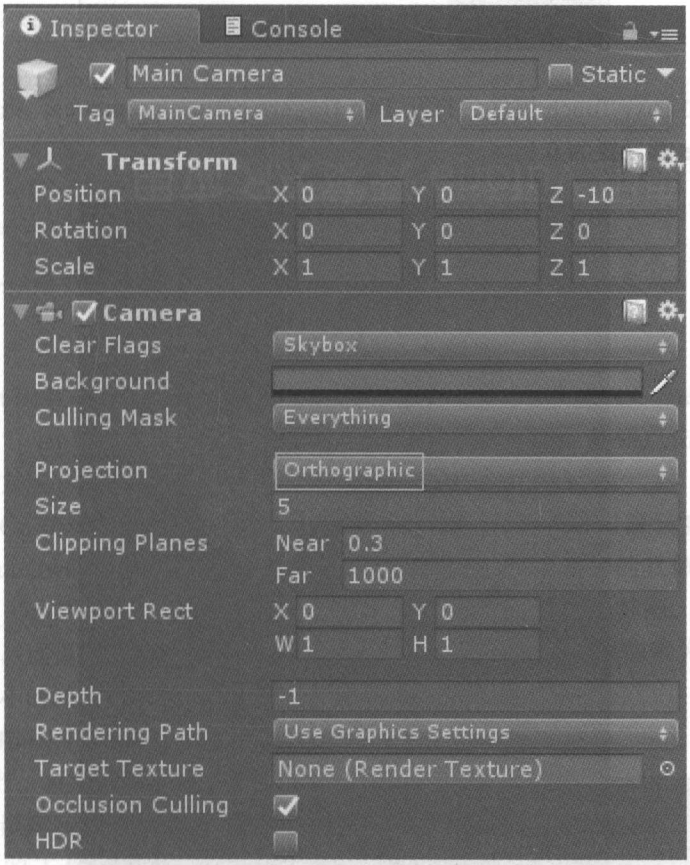
可以通过下面的例子来了解如何使用 Easy Touch 识别相机中两个手指的放大缩小和旋转屏幕的功能。

1）导入 Easy Touch 插件。

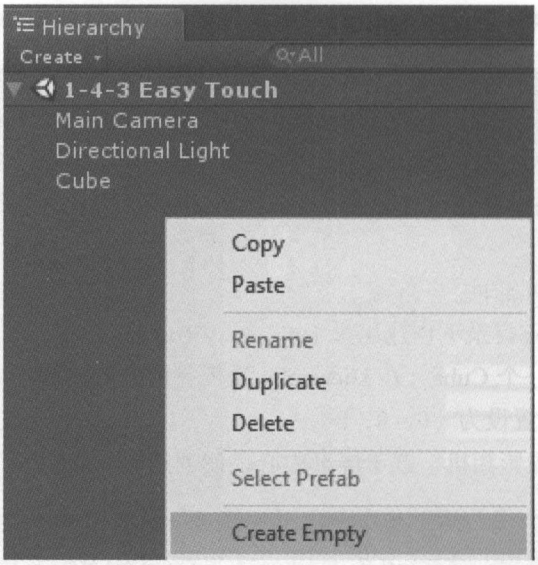
2）在 Assets/Scenes 目录下创建场景 1-4-3 Easy Touch。

3）在场景中创建一个 Cube，在 Hierarchy 面板空白处点击鼠标右键，依次选择 3D Object → Cube，并将其位置设为（0，0，0）。

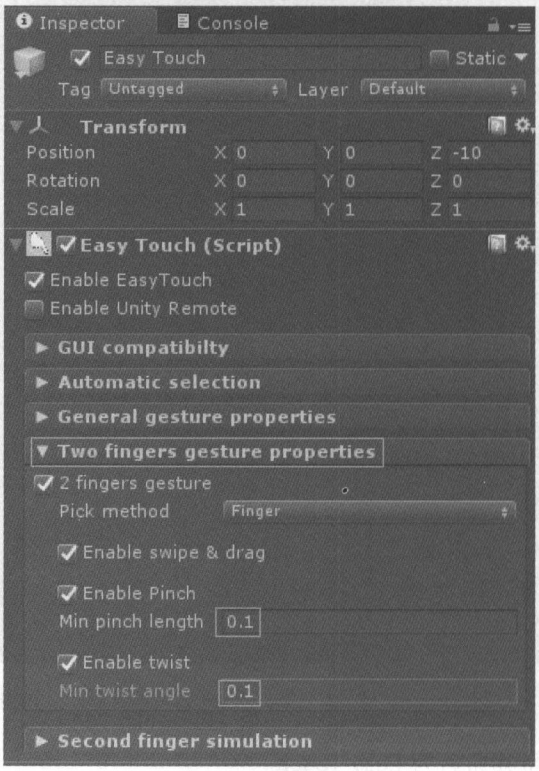
4）将相机设为正投影相机，置于屏幕中心，使创建的 Cube 位于视野中心。



5）在场景中创建一个空的 GameObject：在 Hierarchy 面板空白处点击鼠标右键，依次选择 Create Empty，并将其命名为 Easy Touch。



6）为其添加 Easy Touch 组件，打开 Two fingers gesture properties，将 Min pinch length 和 Min twist angle 都设置为 0.1。



7）在 Assets/Scripts 目录下创建脚本 EasyTouchTest.cs，输入以下代码：

```
using UnityEngine;
using HedgehogTeam.EasyTouch;

public class EasyTouchTest : MonoBehaviour
{
    Camera controlledCamera;

    void Start()
    {
        controlledCamera = GetComponent<Camera>();
        if (controlledCamera == null)
            Debug.LogError("Controller camera is null!");

        EasyTouch.On_Twist += EasyTouch_On_Twist;
        EasyTouch.On_Pinch += EasyTouch_On_Pinch;
    }

    private void EasyTouch_On_Twist(Gesture gesture)
    {
        Vector3 angle = transform.localEulerAngles;
        angle.z += gesture.twistAngle;
        transform.localEulerAngles = angle;
    }

    private void EasyTouch_On_Pinch(Gesture gesture)
    {
        controlledCamera.orthographicSize += gesture.deltaPinch * 0.01f;
    }
}
```

首先在 Start 函数中使用 GetComponent 方法获取 GameObject 的 Camera 对象，然后注册 EasyTouch 的 Twist（旋转）和 Pinch（缩放）事件，类似于注册按钮点击事件，事件发生时会调用注册的方法。

EasyTouch_On_Twist 会在 Easy Touch 检测到旋转时调用，这时旋转自身 Transform 相应的角度（Gesture.twistAngle 就是 Easy Touch 在检测到旋转时传入的具体角度数据，有正负）。EasyTouch_On_Pinch 也类似，这里我们调整了 Camera 的视野范围，在屏幕上就会有拉近拉远的感觉。这里的移动量乘以了 0.01，因为这里的移动量是以像素为单位的，直接使用会使放大的缩小的幅度过大。

8）将 EasyTouchTest.cs 拖至 Main Camera。

9）点击运行，在 PC 上可以按住 Alt 键模拟多点触控。尝试旋转，缩放。

Easy Touch 的工作模式如上所述：注册指定的事件，在响应方法中处理自己的业务逻辑。Easy Touch 还提供了简单组件可以直接进行简单的操作，例如缩放、移动等。具体操作方式可以参考 API 文档。

Chapter 2 第 2 章

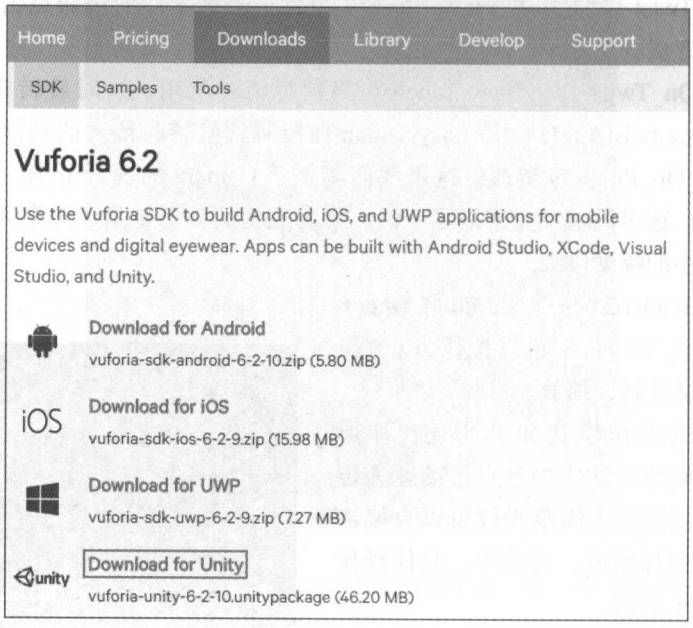
使用 Vuforia 开发 AR 应用

2.1 Vuforia SDK 简介

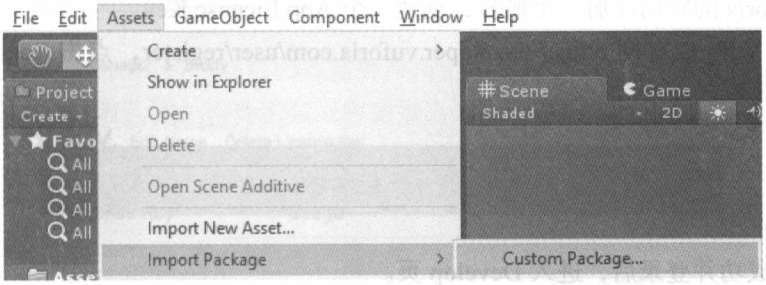
Vuforia SDK 是高通公司针对移动平台推出的增强现实开发包，开发者可以使用该 SDK 便利地实现图像和物体，并添加自定义功能和内容来开发 AR 应用。

2.1.1 Vuforia SDK 的下载与安装

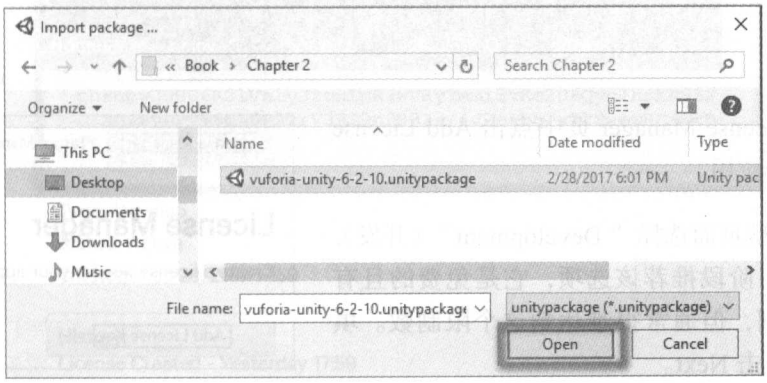
1）登录 Vuforia 的官方 SDK 下载地址，下载最新版本的 Unity SDK，下载地址：https://developer.vuforia.com/downloads/sdk。



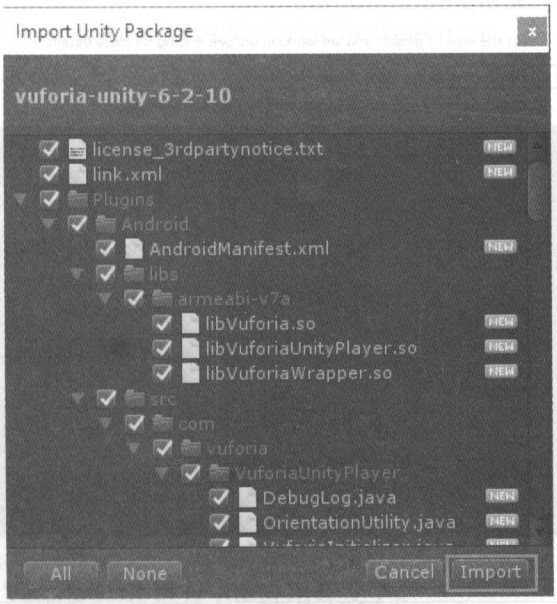
2）下载完成后，创建项目 Vuforia Samples，在菜单栏点击 Assets → Import Package → Custom Package。



3）选择下载的文件（本书版本为 6.2.10），点击 Open。



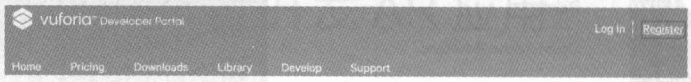
4）点击 Import。



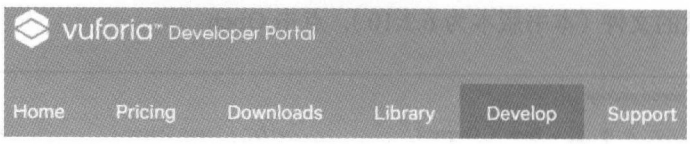
2.1.2 创建 App License Key

使用 Vuforia 开发需要一个 App License Key 才能使 SDK 正常工作，因此在开发前，我们需要在 Vuforia 的官网注册一个账号，创建一个 App License Key。

1）登录 Vuforia 官网 https://developer.vuforia.com/user/register，点击 Register 注册一个免费账号。

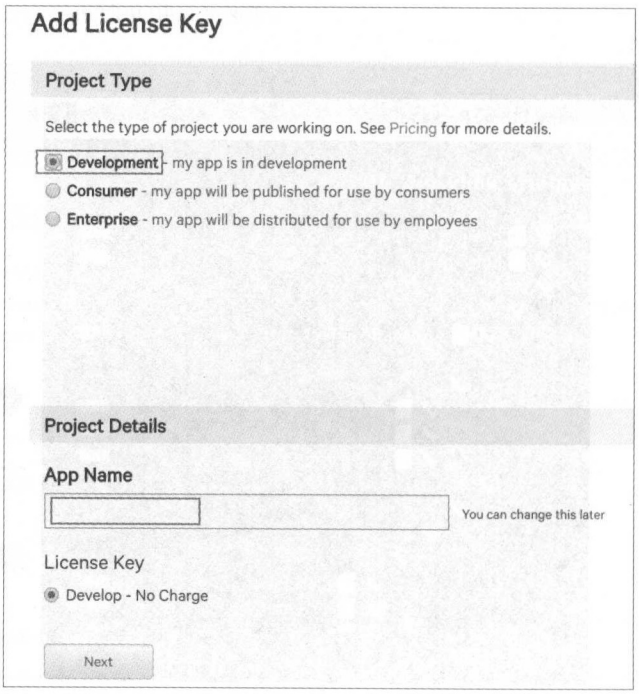


2）注册成功并登录后，进入 Develop 页。

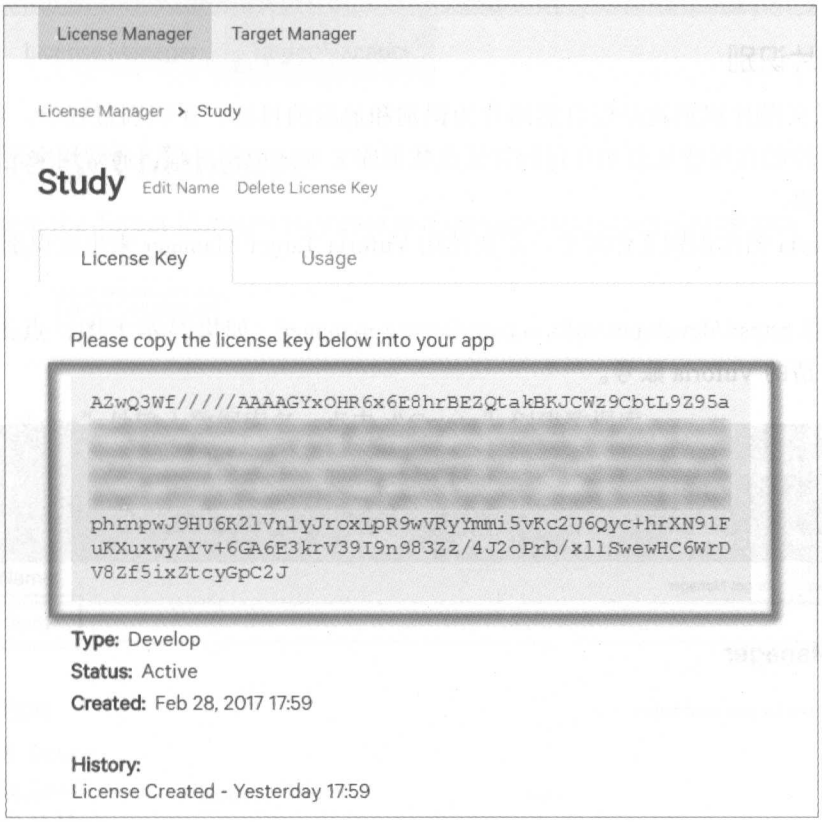


3）在 License Manager 页中点击 Add License Key 创建一个应用。

4）在类型页面选择“Development”（开发），在开发或学习阶段推荐该选项，它是免费的且有使用数量限制，但通常很难达到这个限制数。填入应用名，点击 Next。

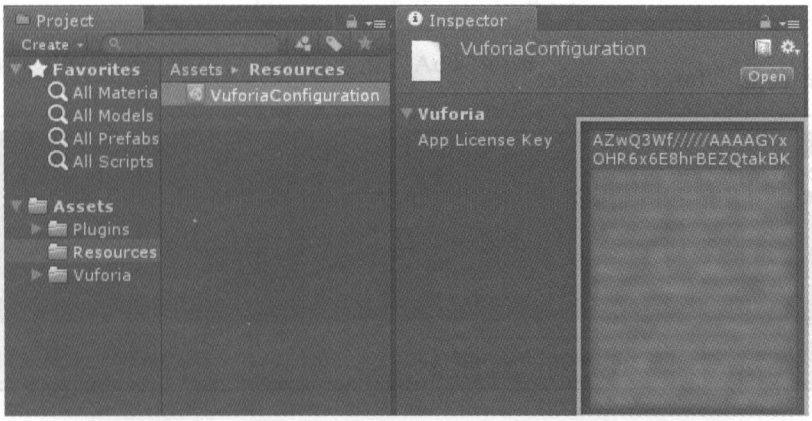


5）随后复制图中的 License Key。



2.1.3　在 SDK 中输入 App License Key

打开 Unity，选中 Assets/Resources/VuforiaConfiguration，在 Inspector 面板中填入刚才复制的 App License Key。



至此就可以使用 Vuforia SDK 进行 AR 应用开发了。

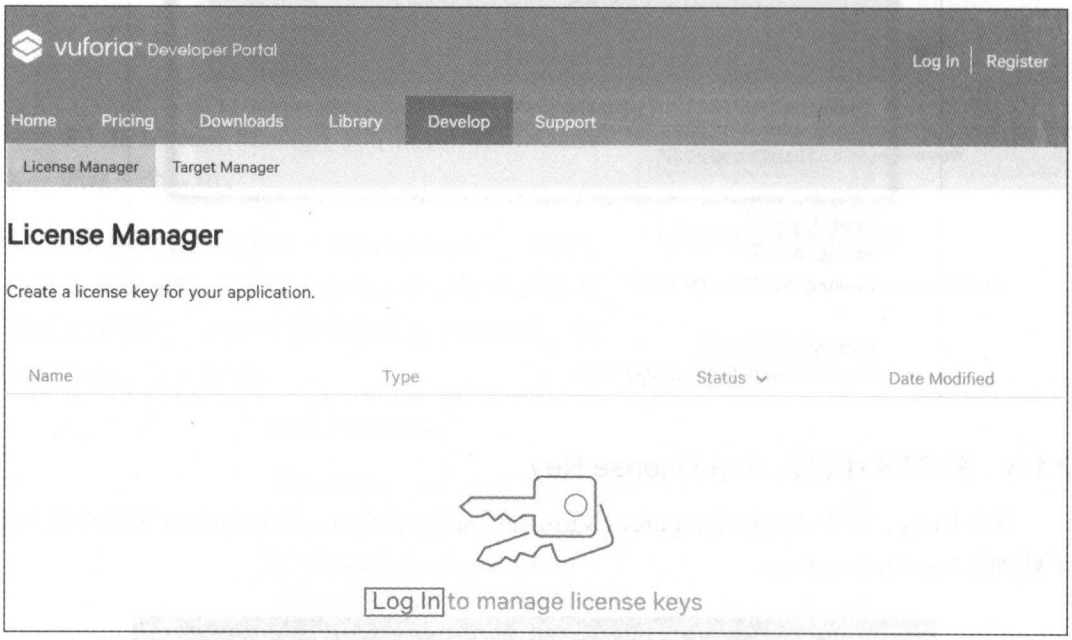
2.2 使用 Vuforia SDK 进行物体识别

2.2.1 图片识别

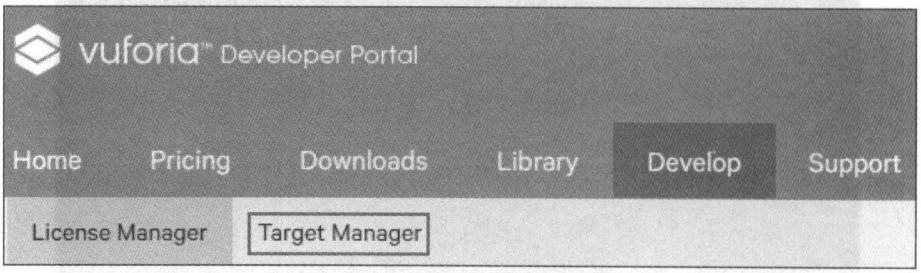
顾名思义图片识别就是以自然图片为识别和追踪的目标。在识别过程中，Vuforia 通过对比输入图像的自然特征点和自身的特征点数据库来确定识别过程，通常用来增强一些打印的图片的效果。

在 Vuforia 图片识别工作流中，需要使用 Vuforia Target Manager 来生成识别图的特征点数据库。

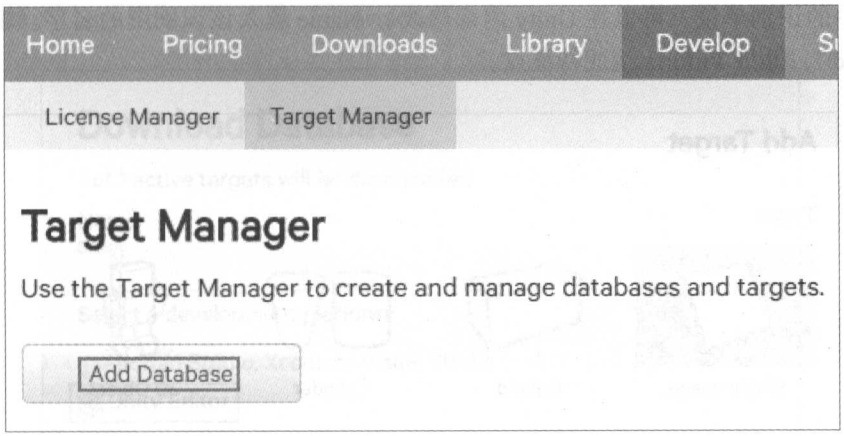
1）登录 https://developer.vuforia.com/license-manager，如果显示下图，点击“Log In”登录刚才申请的 Vuforia 账号。



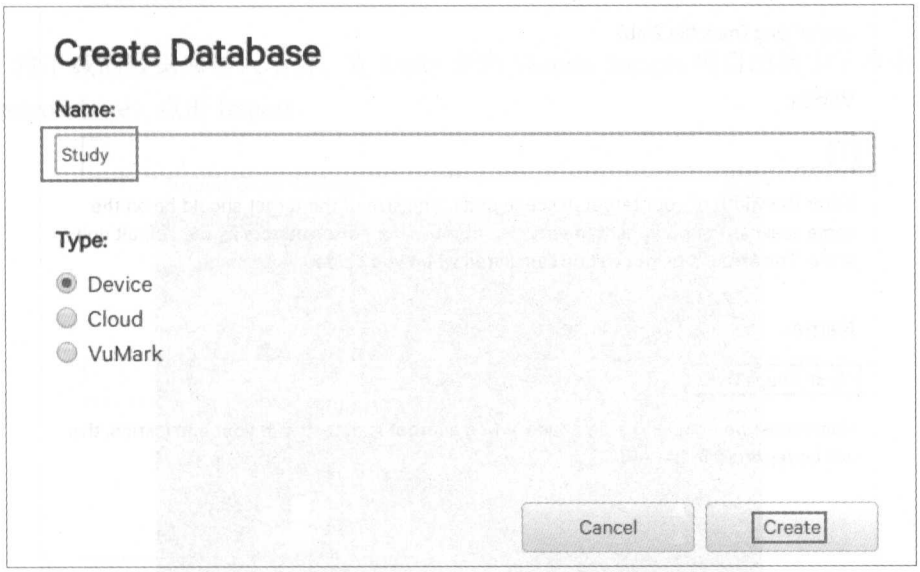
2）登后点击“Target Manager”。



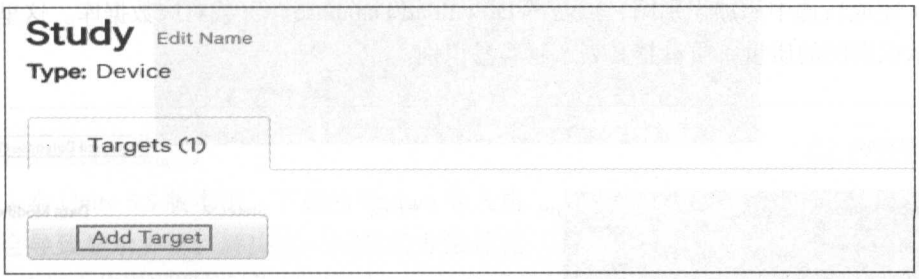
3）点击“Add Database”创建一个特征点名为 Study 的识别数据库。



4）在“Name”处输入数据库名，点击“Create”创建数据库。

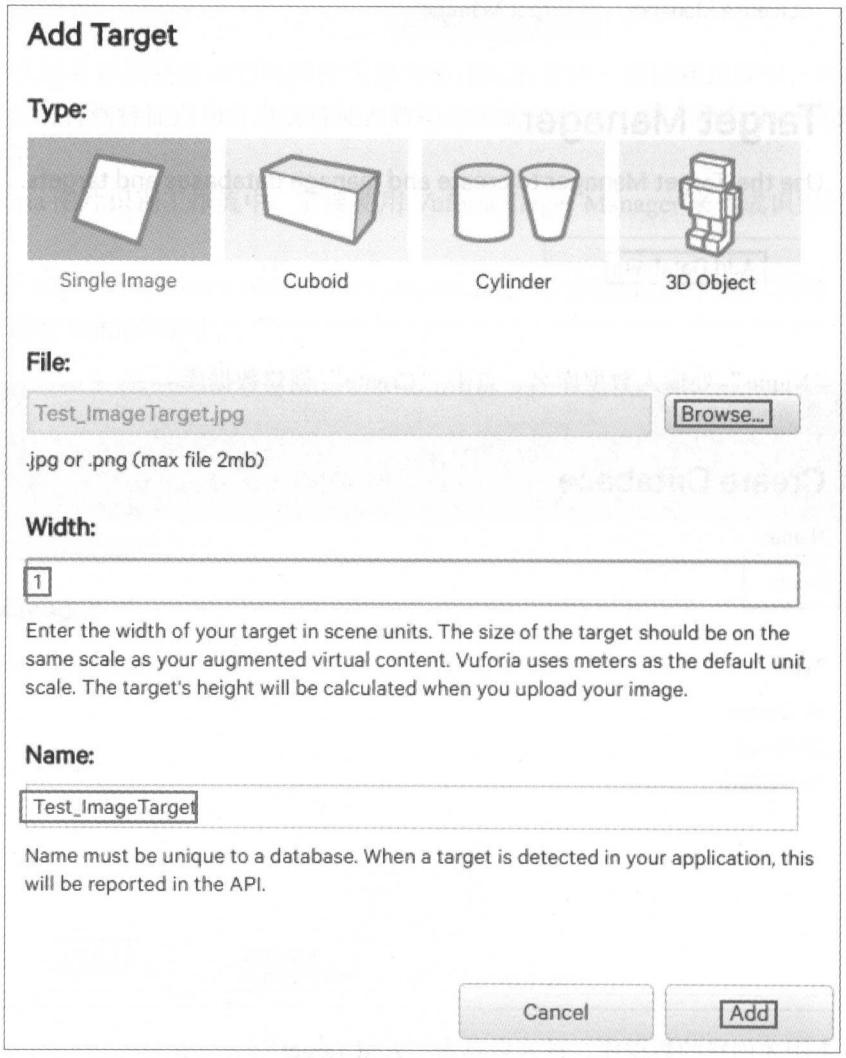


5）点击刚才创建的数据库，进入后点击“Add Target”。

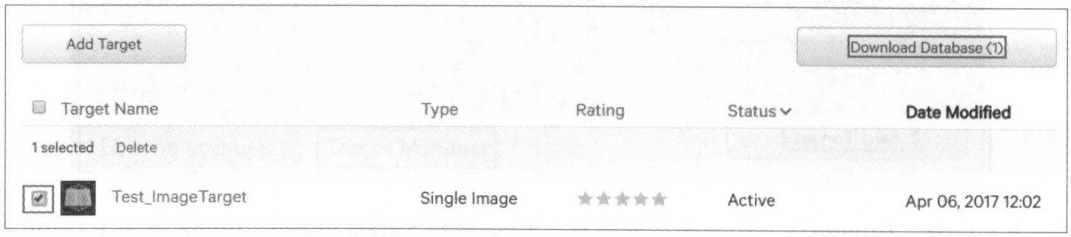


6）点击“Browse”上传一张本地图片，Type 选择 Single Image，Width 填入 1（这里对

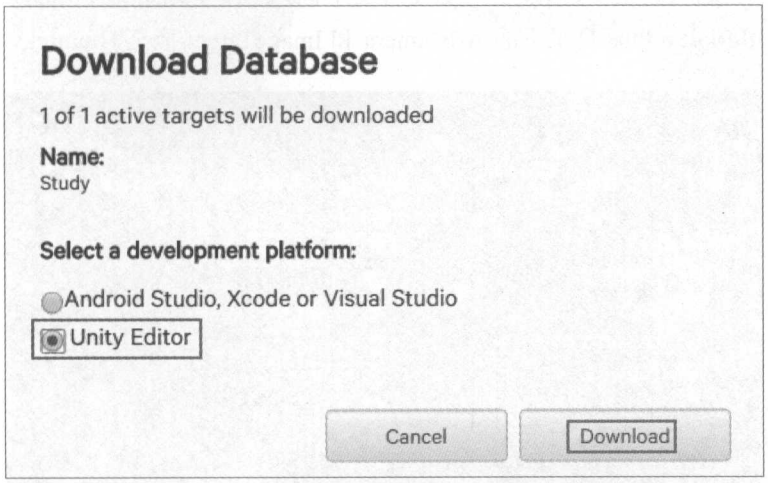
应 Unity 中的识别单位，可以在 Unity 内部修改），Name 填入该识别图的名字。填写完毕后点击“Add”，等待 Vuforia 生成特征点。



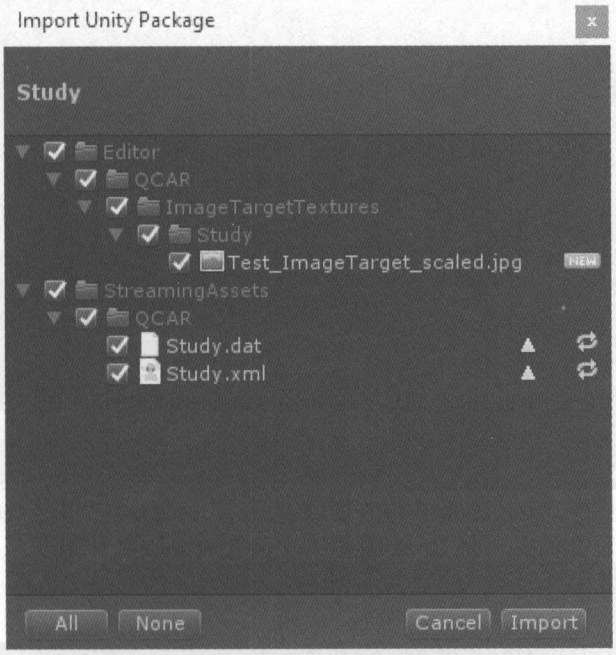
7）完成后选中添加识别图，点击“Download Database”下载对应数据库。这里的 Rating 表示识别图的质量，质量越高表示越容易识别。



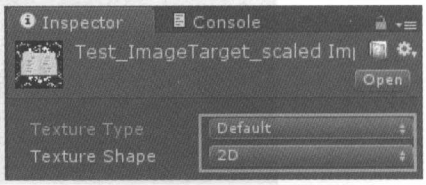
8）选择“Unity Editor”，点击“Download”。



9）将下载的数据库导入项目，在 Unity 开启 Vuforia Sample 项目的状态下双击下载的 Study.unitypackage，点击 Import。



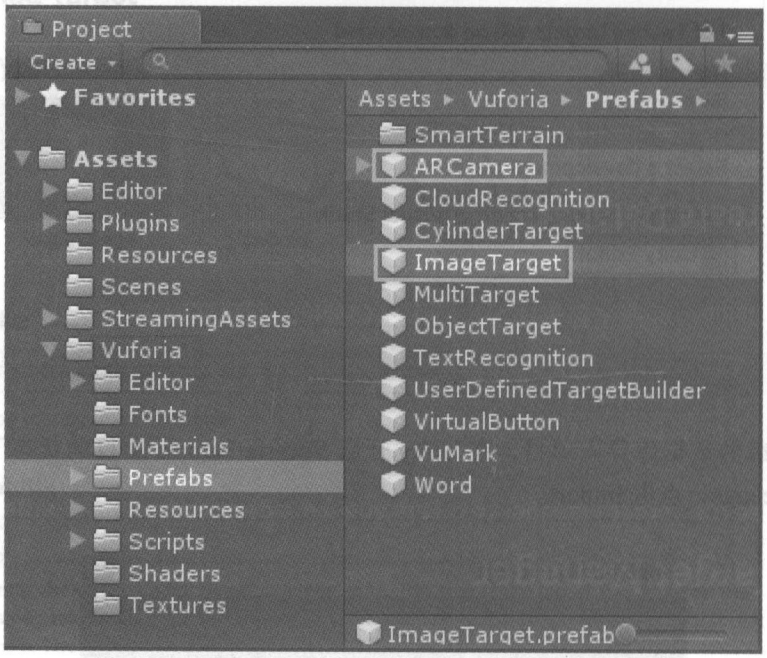
10）在 Unity 5.5 版本中，下载的 Texture 导入格式错误会导致显示异常，解决这一问题的方法是在 Inspector 面板中将识别图的贴图格式设为 Normal。例如 Assets/Editor/QCAR/ImageTargetTextures/Study/



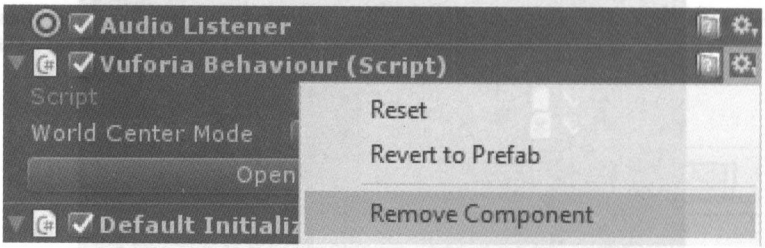
Test_ImageTarget_scaled 文件，在 Inspector 中将其 Type 设置为 Default，Shape 设为 2D。

11）创建 Assets/Scenes 文件夹，在该文件夹下创建一个空场景 Test Image Target。

12）将 Vuforia/Prefabs 目录下的 ARCamera 和 ImageTarget 拖至 Hierarchy 面板。

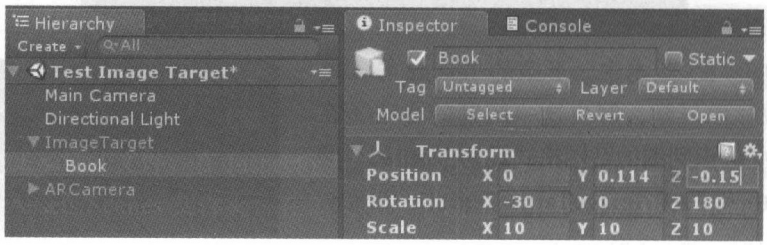


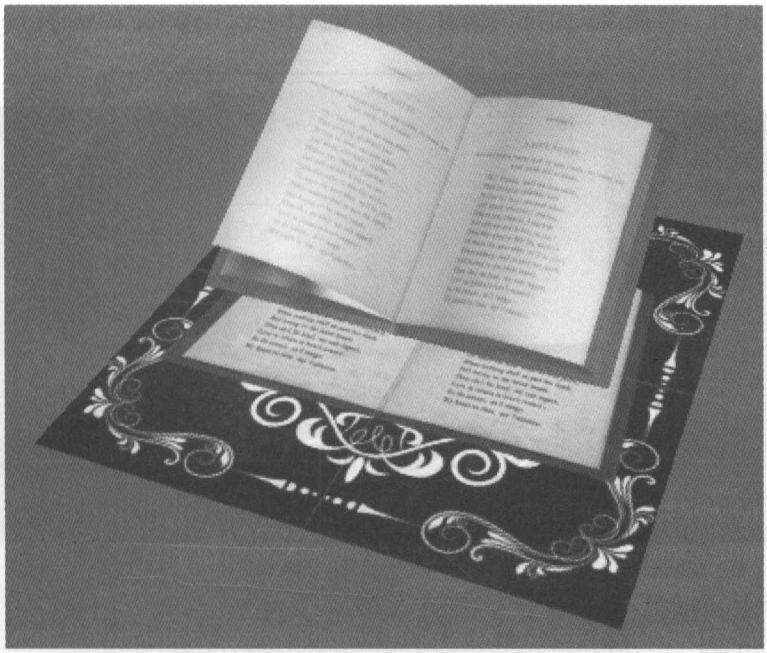
13）选中 ARCamera，在 Inspector 面板中删除 ARCamera 的 Audio Listener 组件（点击右上角的小齿轮）。



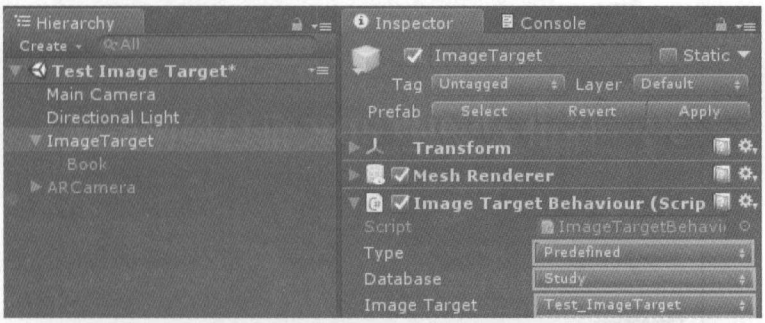
14）创建 Models 文件夹，将 Book 模型的文件夹放入该文件夹下。

15）将 Book 模型拖至 ImageTarget 下并调整其尺寸。

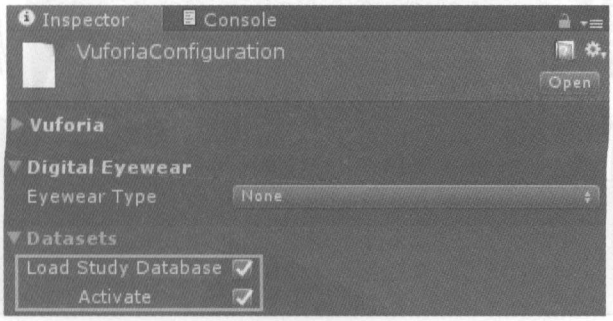




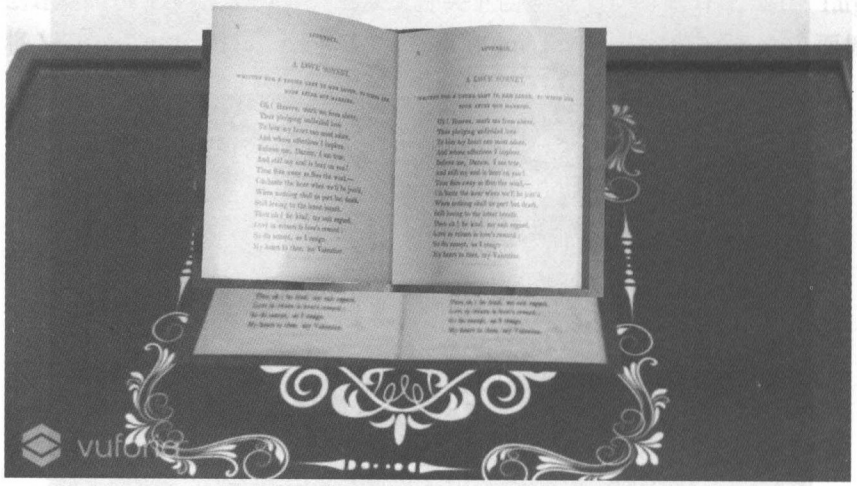
16）选中 ImageTarget，在 Inspector 面板中将 Database 选为 Study，Image Target 选为 Test_ImageTarget，Type 选为 Predefined。



17）选中 Assets/Resources/VuforiaConfiguration，在 Inspector 面板的 Datasets 中选中“Load Study Database”和“Activate”，在运行时加载并激活 Study 数据库。



18）点击运行，将摄像头对准识别图，可以看到 ImageTarget 下的物体。

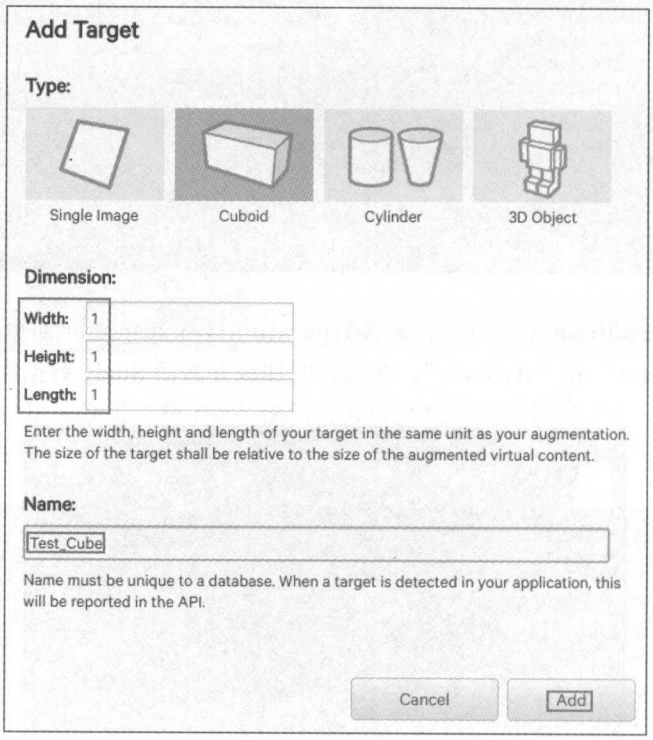


2.2.2 长方体识别

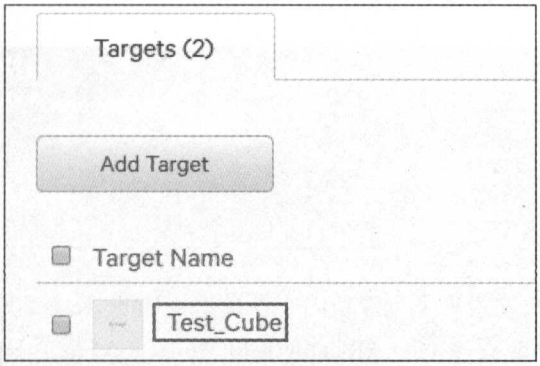
Vuforia 也支持长方体识别，工作流类似于图片识别。

1）进行图片识别的 1 ~ 5 步的操作，进入开发者中心的“Target Manager”，点击 Add。

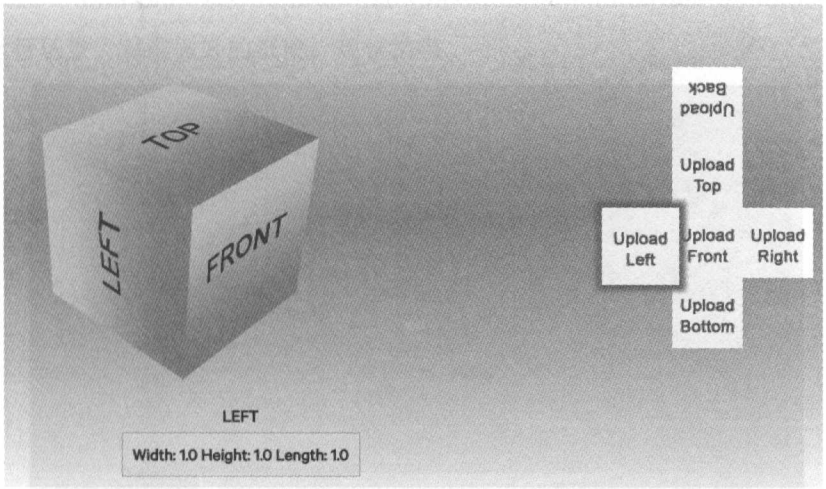
2）Type 选择 Cuboid，在 Dimension 中填入长方体的宽、高、长、输入名称，点击“Add”。



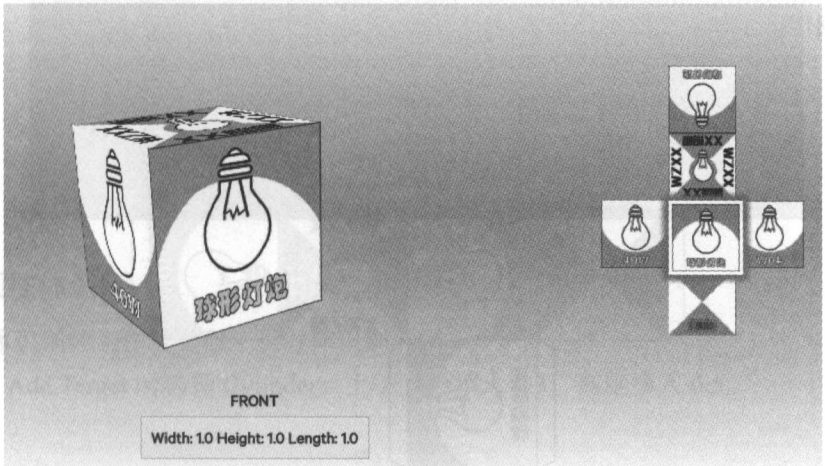
3）在 Target 列表中选择 Test_Cube，进入其属性编辑界面。



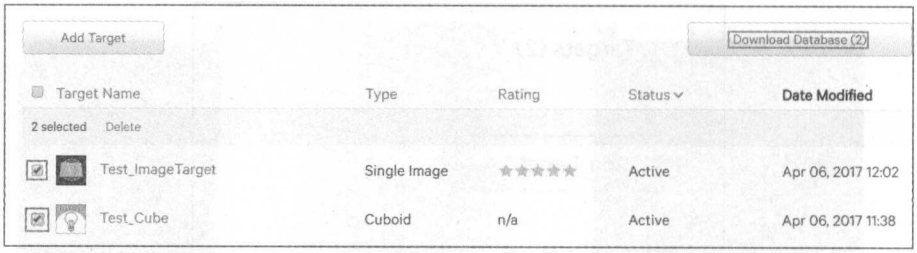
4）在右侧的面板选择面板中选择对应面，上传上、下、左、右、前、后六个面的图片。



5）上传完成后结果如下图所示。



6）执行图片识别的 7 ~ 9 步的操作，更新 Vuforia 识别图数据库。

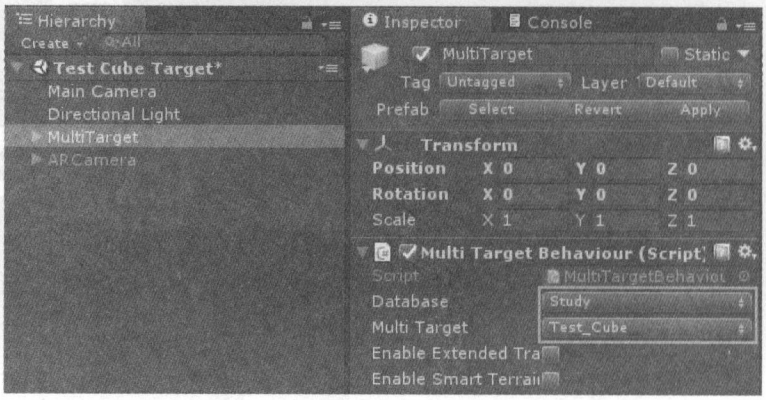


7）在 Assets/Scenes 文件夹下创建新的空场景 Test Cube Target。

8）将 Vuforia/Prefabs 目录下的 ARCamera 和 MultiTarget 拖至 Hierarchy 面板。

9）删除 ARCamera 上的 Audio Listener。

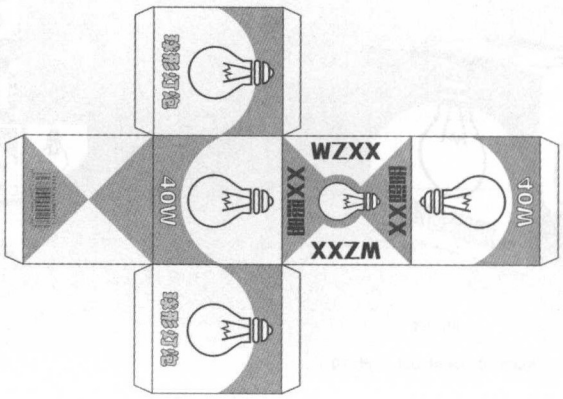
10）选中 MultiTarget，在 Inspector 面板中将 Database 设为 Study，将 Multi Target 设为 Test_Cube。



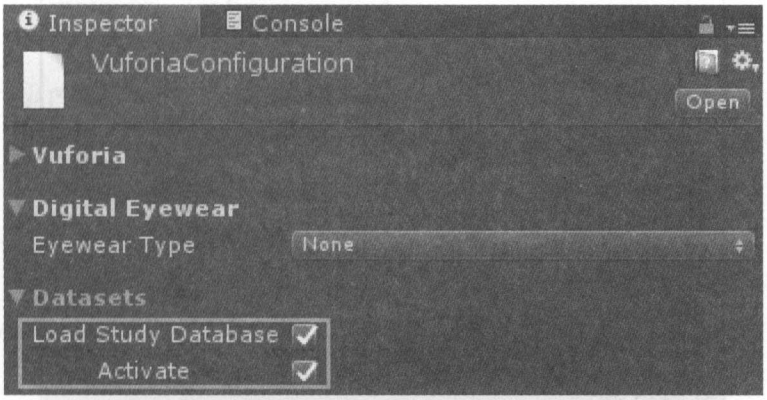
11）导入 Particle.unitypackage。

12）将导入的 Prefabs/Particle Effect 拖至 MultiTarget 下并调整位置。

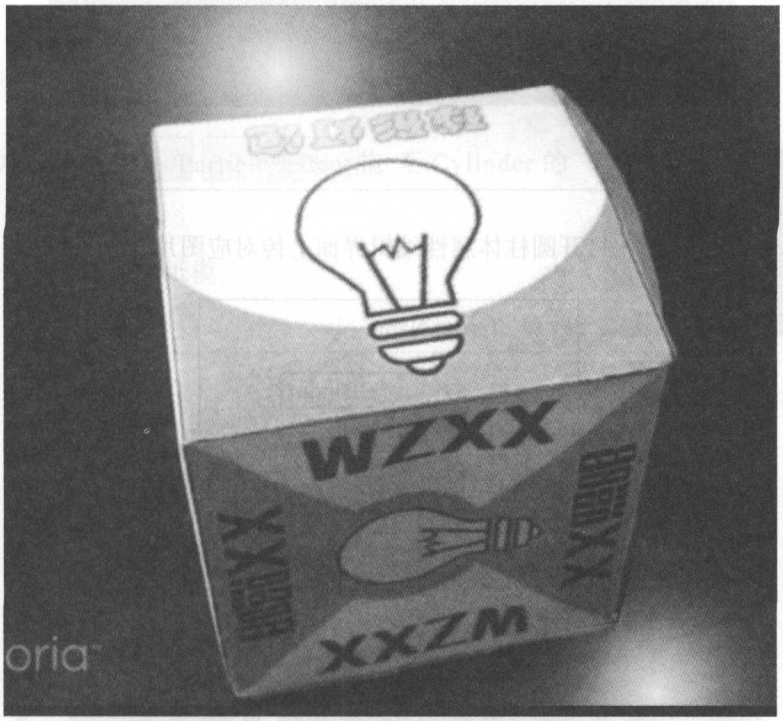
13）制作一个如下图所示的正方体的纸模。



14）确保 Assets/Resources/VuforiaConfiguration 的 Dataset 中的 Study 数据库被加载且被激活。



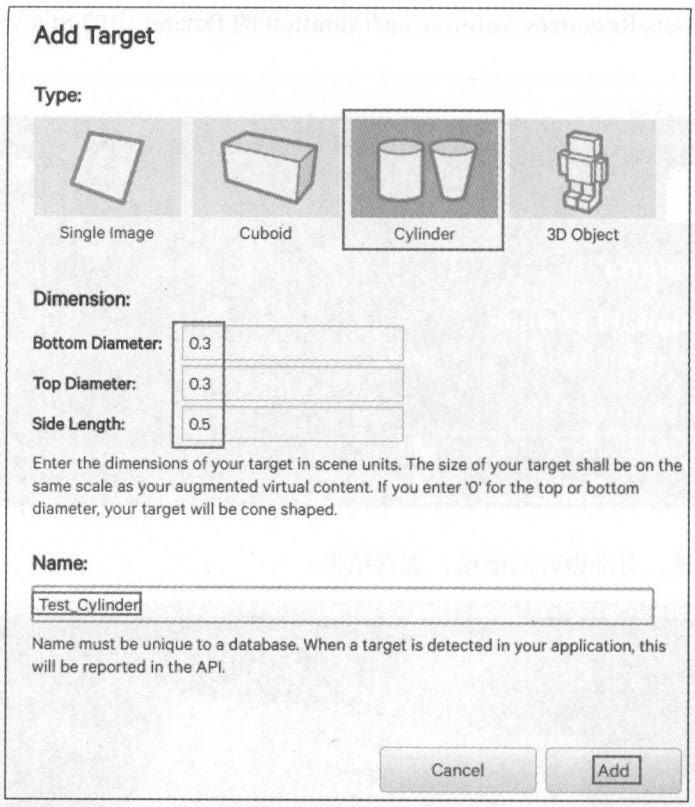
15）运行场景，对准做好的纸模，观察结果。



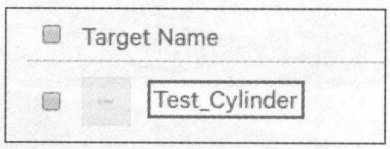
2.2.3　圆柱体识别

圆柱体识别的工作流类似于长方体识别。

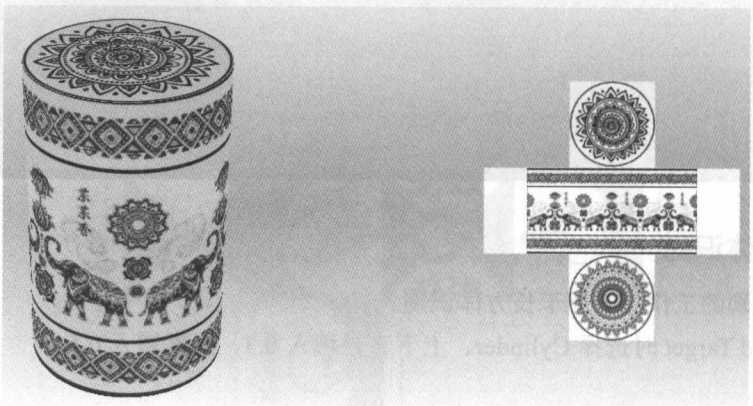
1）在 Add Target 时选择 Cylinder，上下直径填入 0.3，高度填入 0.5。



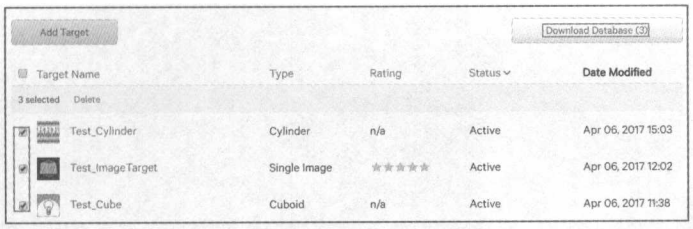
2）点击 Test_Cylinder 打开圆柱体属性编辑界面上传对应图片。



3）上传上面、下面、侧面的图片。



4）下载识别数据库并导入。

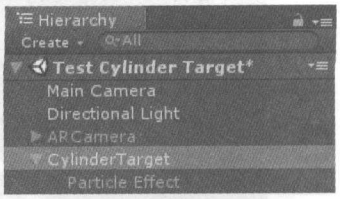


5）将 Test Cube Target 场景在同一目录下另存为 Test Cylinder Target。

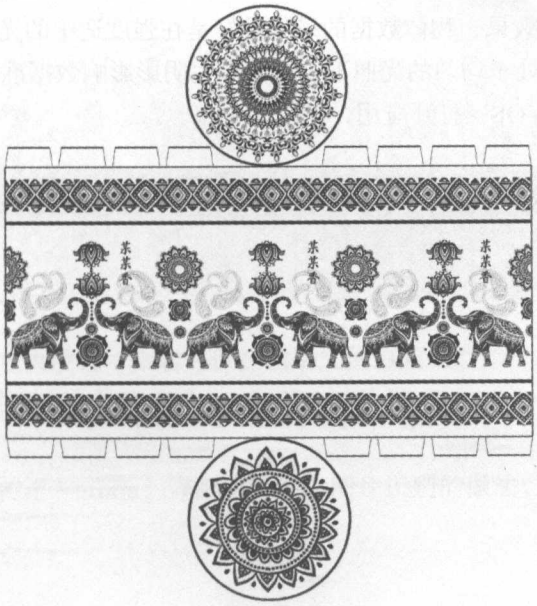
6）删除 MultiTarget，将 Assets/Vuforia/Prefabs/CylinderTarget 拖至 Hierarchy 面板。

7）将 CylinderTarget 的 Database 设为 Study，Cylinder Target 设为 Test_Cylinder。

8）将 Assets/Prefabs/Particle Effect 拖至 Cylinder 的下面。



9）根据纸模图纸制作纸模。



10）点击运行，对准做好的纸模，观察结果。

2.2.4 物体识别

物体识别和传统图像识别不同，因为物体的二维图像在不同角度的变化是和 2D 图像不同的。因此 Vuforia 推出了 Vuforia Object Scanner（以下简称 VOS）工具来获取物体数据（Object Data，也就是 .OD 文件），利用该文件进行物体识别。

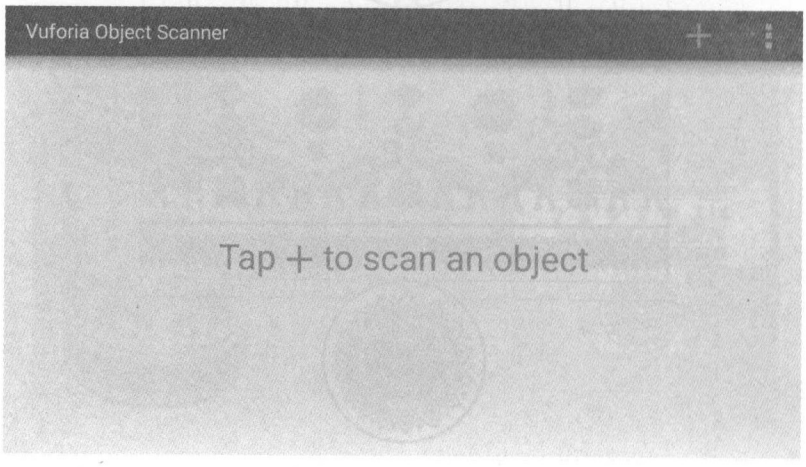
首先我们需要使用 VOS 采集物体的数据。

1）下载并安装 VOS，下载地址为 https://developer.vuforia.com/downloads/tool?d=windows-36-17-4697。

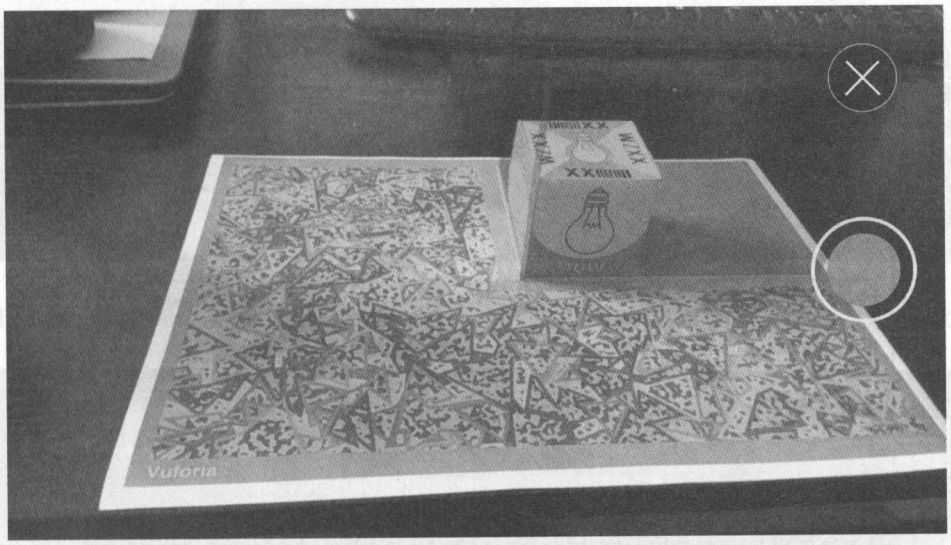
官网有提及在三星 Galaxy S6 和 Galaxy S7 测试会获得最好的识别效果，由于摄像头质量的差异，会导致每个设备采集的数据都有偏差。

为了获得最理想的效果，物体数据的采集最好是在强度适中的光线条件下进行，如果条件允许最好让各表面都处于均匀的光照下，避免产生阴影影响数据准确性。

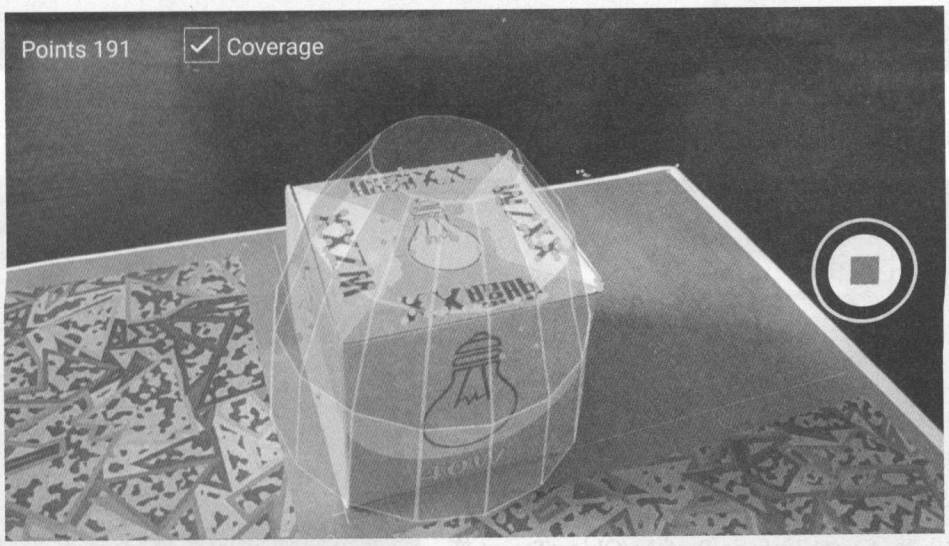
2）在手机上安装 VOS，打开应用，点击“+”。



3）将之前制作的 Box 纸模放在如下图所示的位置。注意，图中坐标轴的原点即扫描物体的原点。



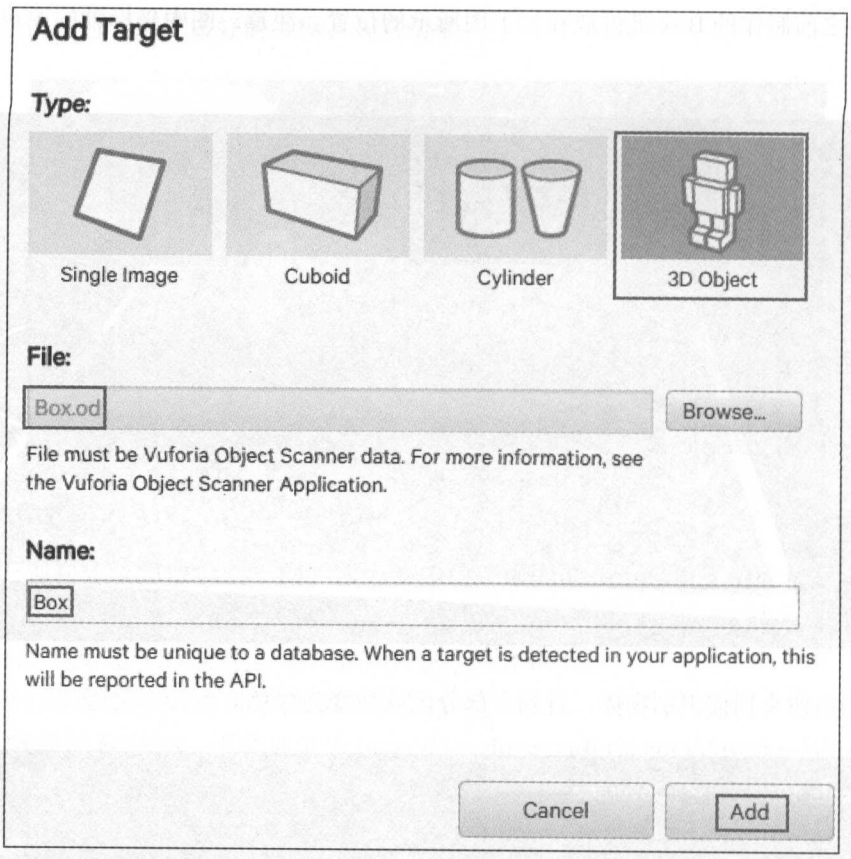
4）用摄像头扫描识别物体，直到大部分区域均变为绿色。



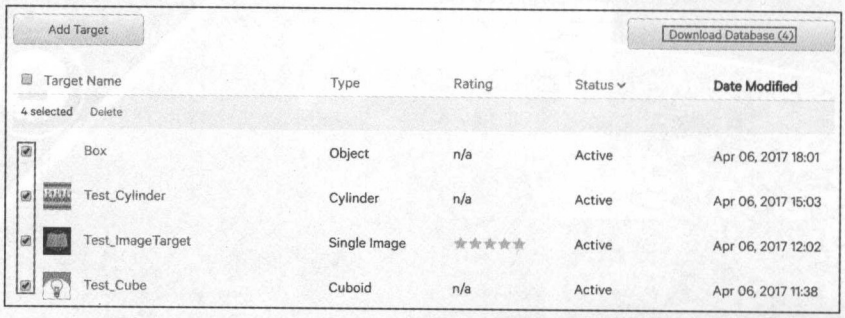
5）点击右侧红色方块按钮，命名并保存识别文件 Box.od。

6）从手机中复制识别文件至本地硬盘（文件路径：/sdcard/ VuforiaObjectScanner/ ObjectReco/ Box.od）。

7）将识别文件上传至 Vuforia，详细做法参考图片识别的做法，但是在 Add Target 那一步选择 3D Object。



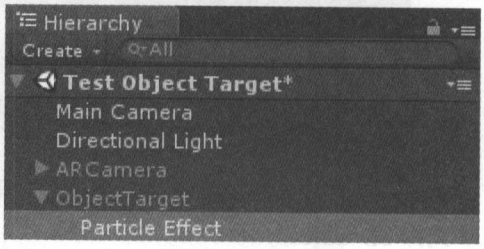
8）下载并更新识别文件数据库。

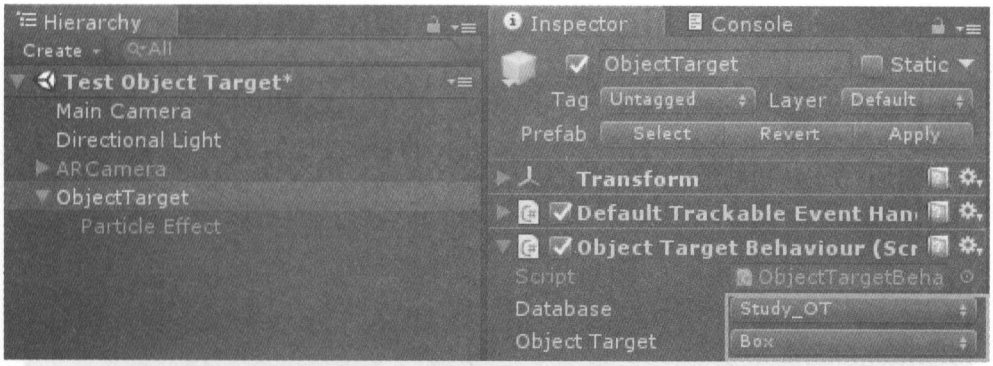


9）将 Assets/Vuforia/Prefabs/ObjectTarget 拖至 Hierarchy 面板。

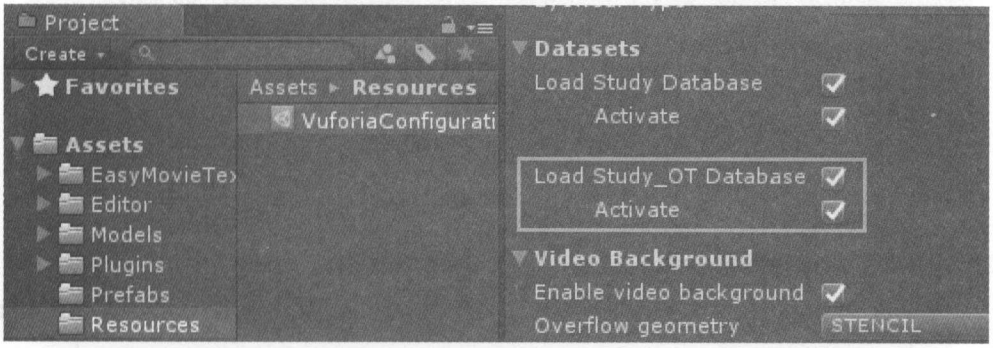
10）将 Assets/Prefabs/Particle Effect 拖至 ObjectTarget 的下面。

11）将 ObjectTarget 的 Database 设为 Study_OT，将 Object Target 设为 Box。

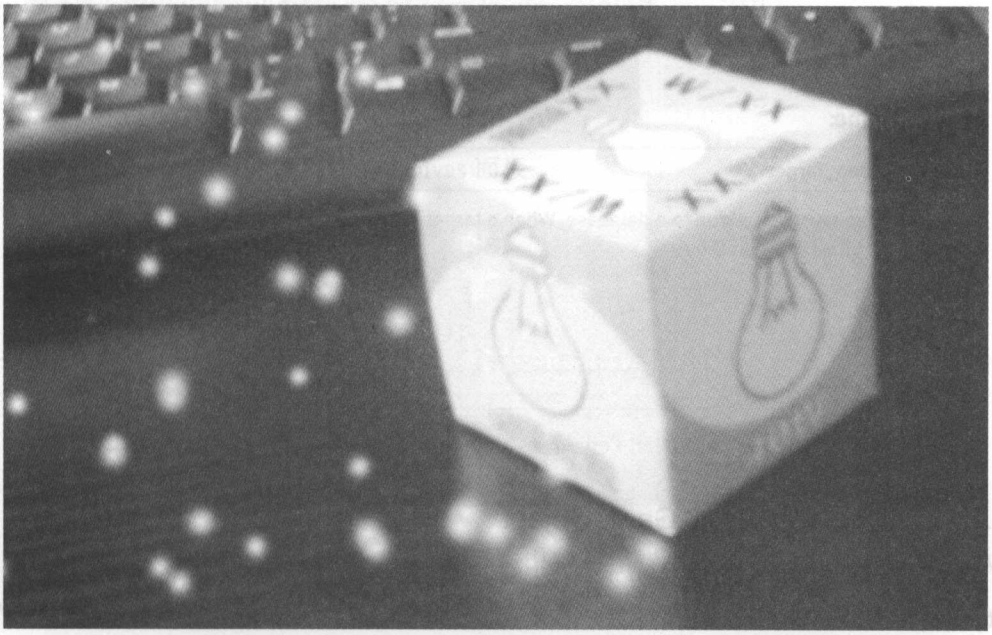




12）确定 Resources/VuforiaConfiguration 中的 Datasets 中的 Study_OT 为选中状态。



13）运行场景，将摄像头对准 Box 的纸模验证结果。

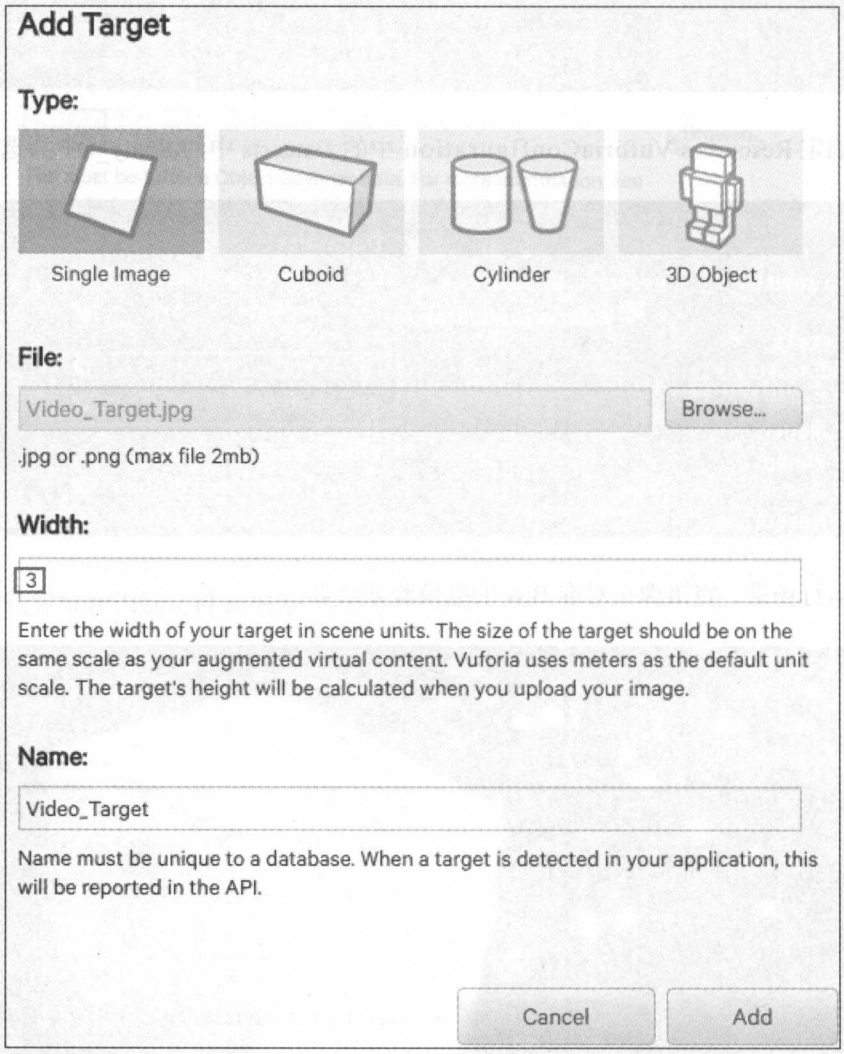


2.3 使用 Vuforia SDK 制作 AR 视频

2.3.1 上传识别图

参照 2.2.1 节上传识别图。

注意，将识别图的宽度设为 3。



上传成功后，下载更新识别图数据库。

2.3.2 创建识别视频播放 GameObject

Easy Movie Texture 是 1.4.1 节介绍过的视频播放插件，本项目将使用 Easy Movie Texture 制作需要的视频播放功能。

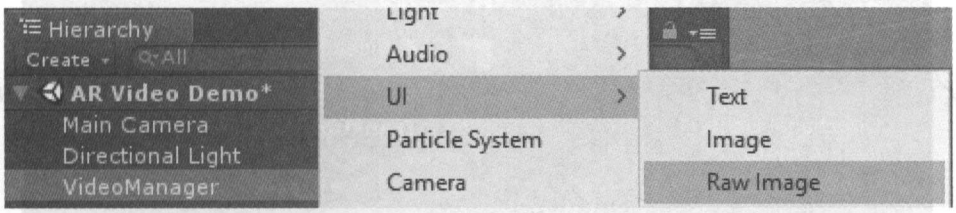
1）在 Unity 的 Asset Store 中下载并导入 Easy Movie Texture（详见 1.4.1 节）

2）在 Assets/Scenes 目录下创建空场景 AR Video Demo。

3）在 Hierarchy 面板中创建一个空的 GameObject，并命名为 Video Manager。

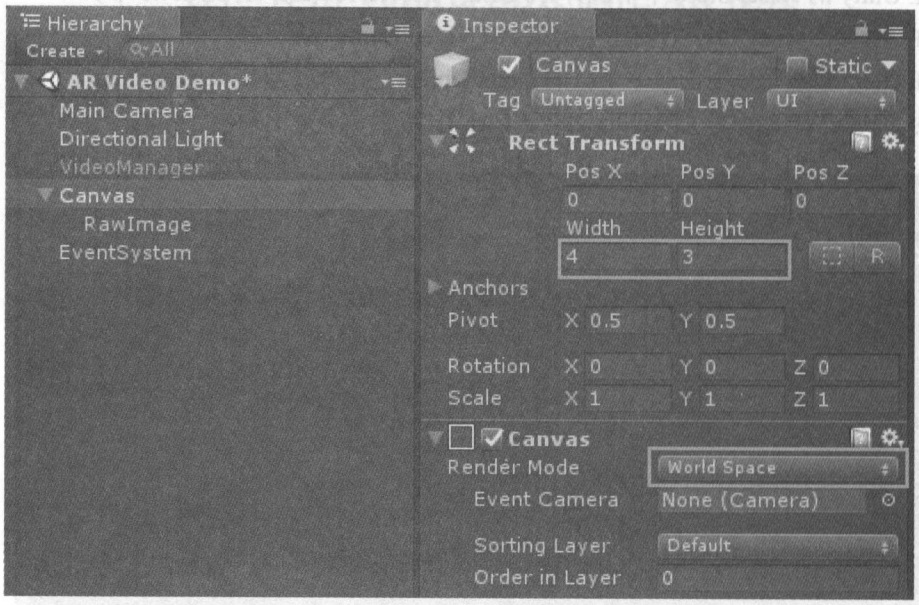
4）为 Video Manager 添加 Media Player Ctrl 组件。

5）创建一个 Raw Image：在 Hierarchy 面板的空白处点击右键，依次选择 UI → Raw Image。



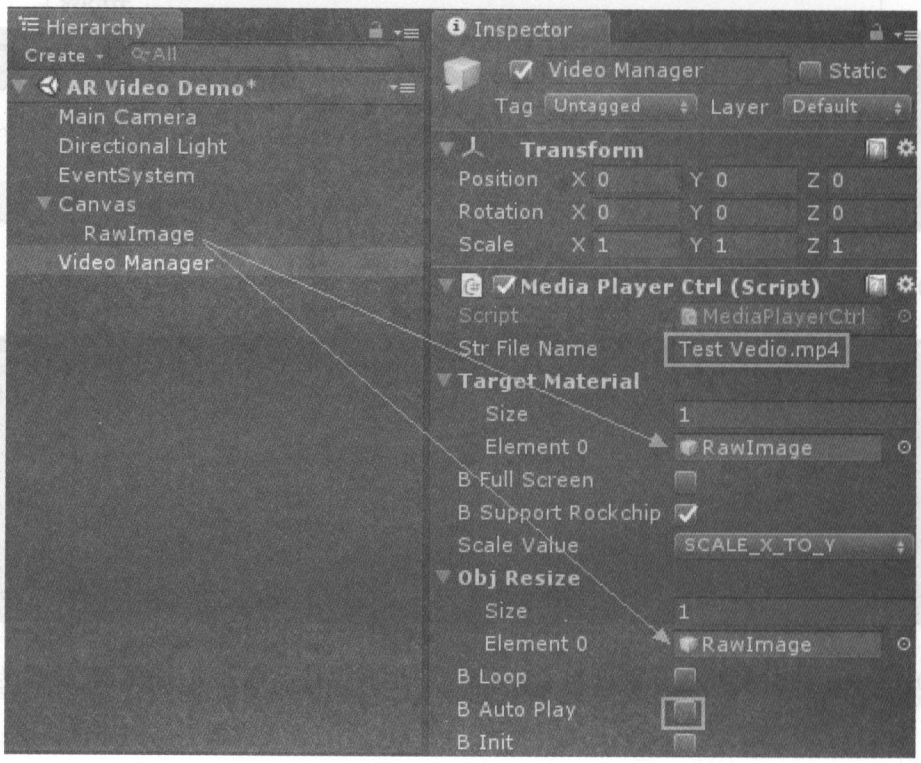
6）将 Raw Image 设为占用整个 Canvas 面板。

7）将 Canvas 的宽和高分别设为 4 和 3（与识别图保持一致），Render Mode 设为 World Space。



8）将测试视频 Test Vedio.mp4 放入 Assets/StreamingAssets 文件夹下。

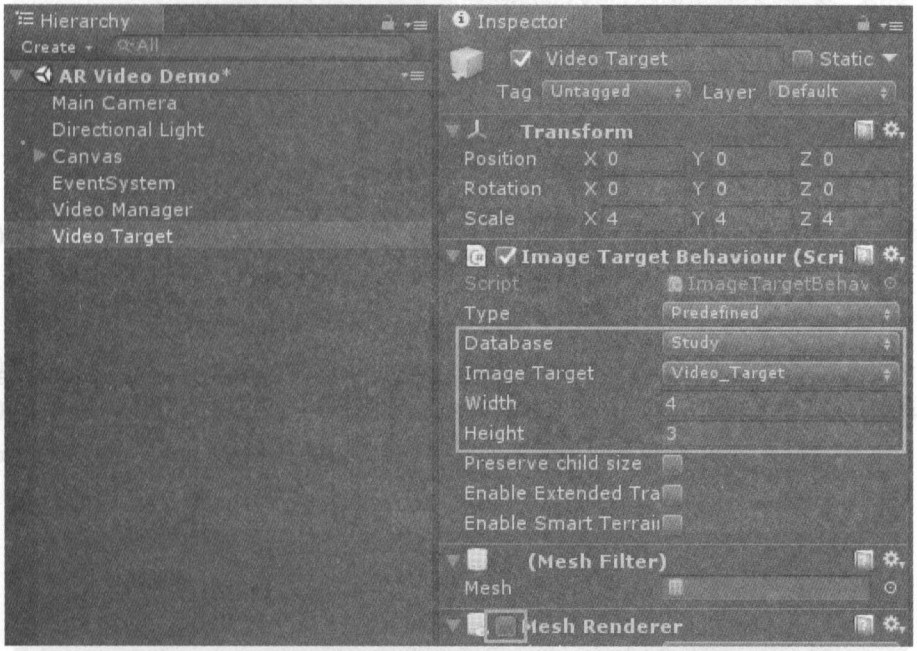
9）将 Raw Image 拖至 Video Manager 的 Media Player Ctrl 的 Target Material 和 Obj Resize 中，在 Str File Name 中填入 Test Vedio.mp4，取消勾选 B Auto Play。



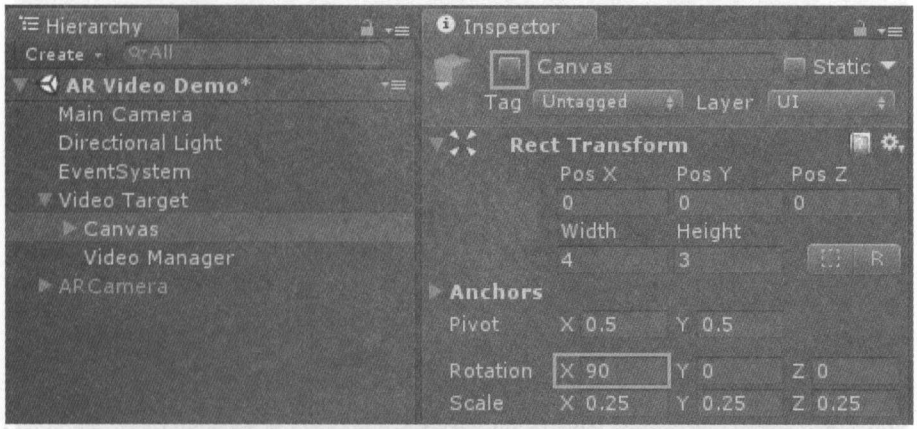
2.3.3　创建识别图目标

1）在 Hierarchy 面板中创建一个空的 GameObject，命名为 Video Target。

2）为其添加 Image Target Behavior 组件，将 Database 设为 Study，将 Image Target 设为 Video_Target，Width 和 Height 分别为 4 和 3，关闭 Mesh Renderer。



3）将 Canvas 和 Video Manager 拖入 Video Target 下，选中 Canvas 的 Rotation 的 X，设为 90 度（注意 Scale 会自动被设为 0.25，保持 0.25 即可）。关闭 GameObject。



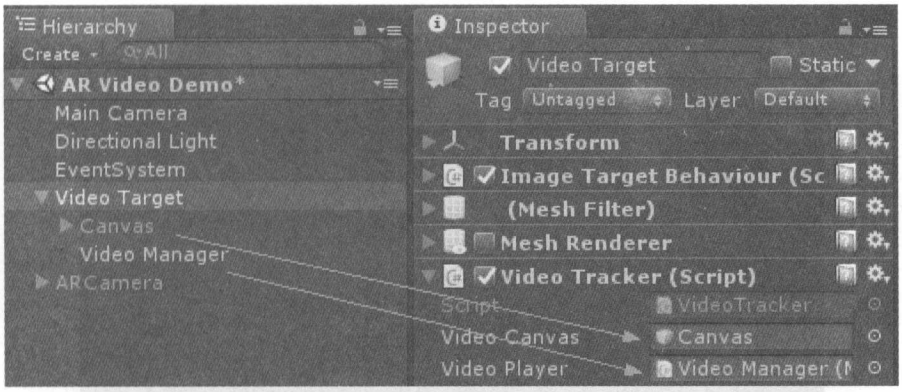
4）在 Assets 目录下创建 Scripts 目录，在该目录下添加 VedioTracker.cs，并添加以下代码：

```
using UnityEngine;
using Vuforia;

public class VideoTracker : MonoBehaviour, ITrackableEventHandler
{
    public GameObject      videoCanvas;
    public MediaPlayerCtrl videoPlayer;

    TrackableBehaviour mTrackableBehaviour;

    void Start()
    {
        mTrackableBehaviour = GetComponent<TrackableBehaviour>();
        if (mTrackableBehaviour) {
            mTrackableBehaviour.RegisterTrackableEventHandler(this);
        }
    }

    public void OnTrackableStateChanged(TrackableBehaviour.Status
previousStatus, TrackableBehaviour.Status newStatus)
    {
        if (newStatus == TrackableBehaviour.Status.DETECTED ||
            newStatus == TrackableBehaviour.Status.TRACKED ||
            newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED) {
            OnTrackingFound();
        } else {
            OnTrackingLost();
        }
    }

    void OnTrackingFound()
    {
        videoCanvas.SetActive(true);
        videoPlayer.Play();
    }

    void OnTrackingLost()
    {
        videoPlayer.Stop();
        videoCanvas.SetActive(false);
    }
}
```

在 Start 方法中先注册 TrackableBehavior 的 Track 事件。

在 Track 事件的回调方法 OnTrackableStateChanged 中当检测到识别图时，调用 OnTrackingFound 方法播放视频，当目标丢失时调用 OnTrackingLost 方法停止播放。

5）选中 Video Target，为其添加 Video Tracker 组件，并将 Canvas 和 Video Manager 的引用赋值。

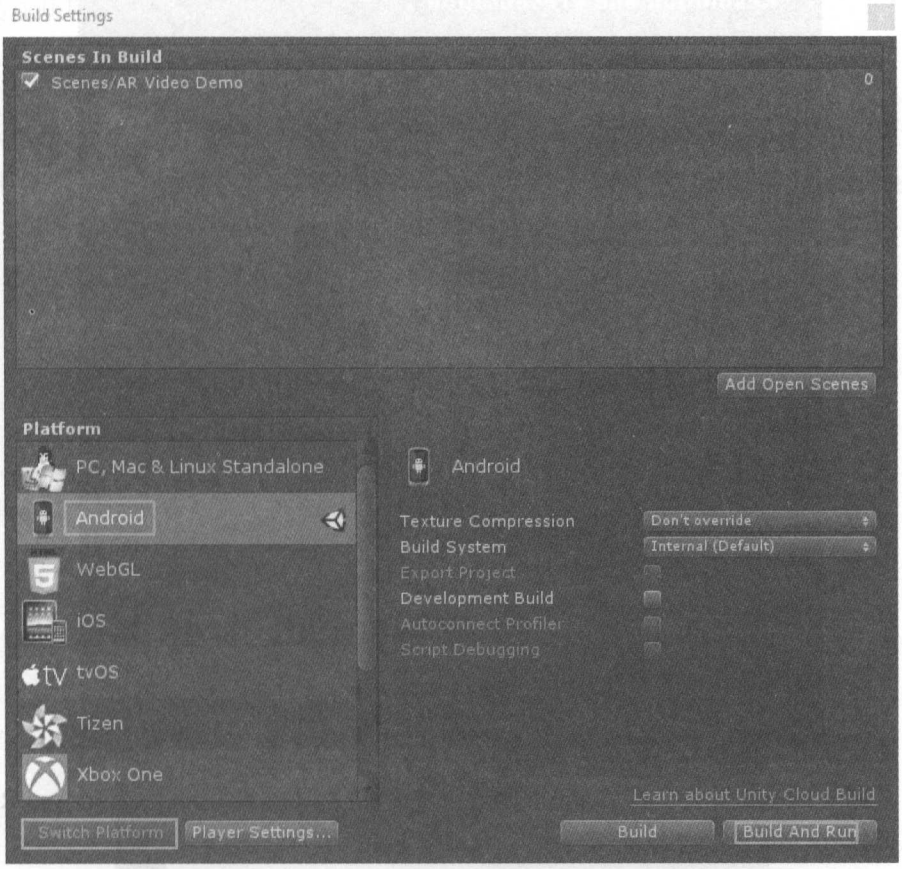


6）将 Assets/Vuforia/ARCamera 拖至 Hierarchy 面板中，删除 Audio Listener 组件。

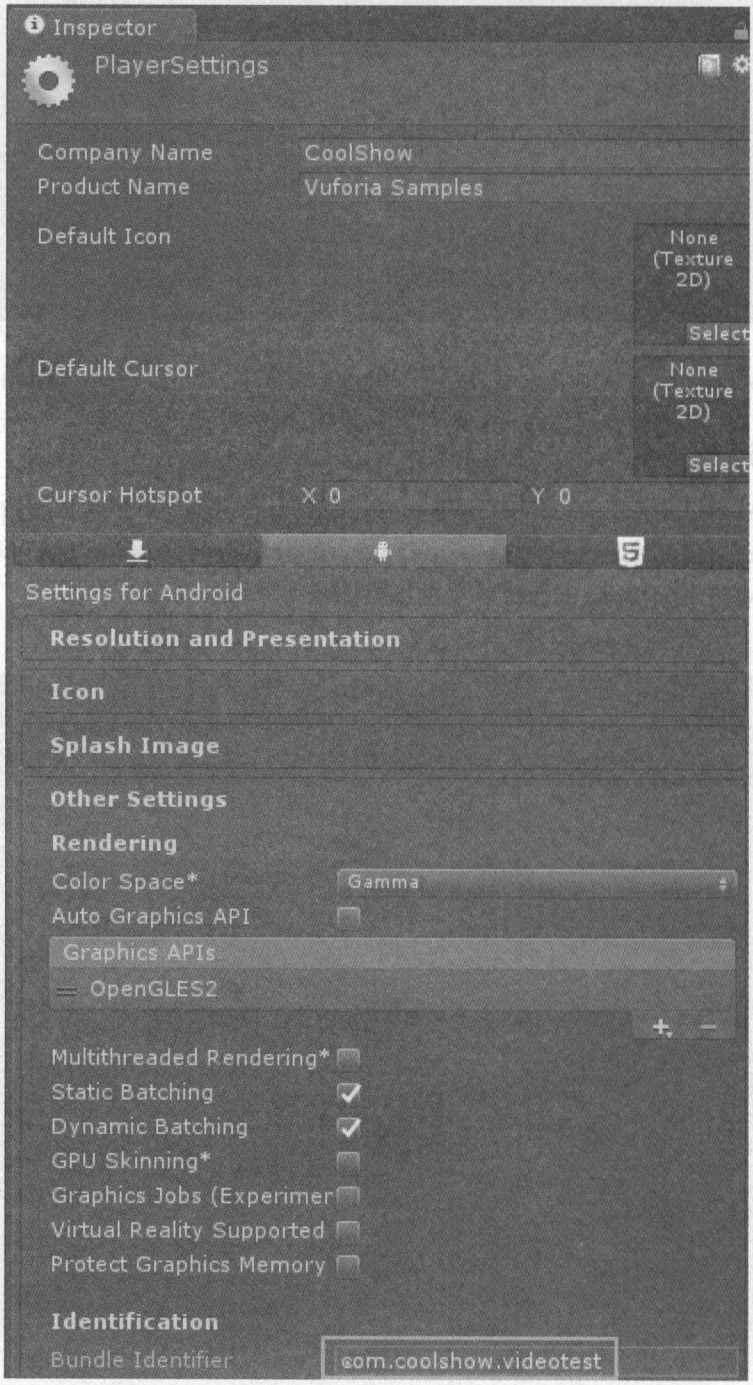
2.3.4　编译运行程序

1）确保 Assets/Resources/VuforiaConfiguration 的 Dataset 中的 Study 数据库为 Active 状态。

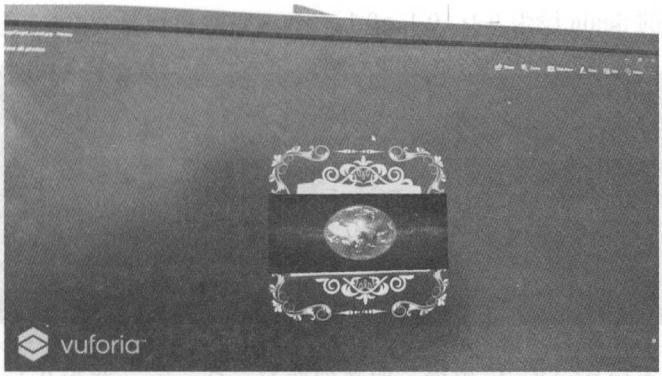
2）在 File → Build Settings 中选中 Android，点击 Switch Platform（确保当前编译环境为 Android），点击 Add Open Scenes 将当前场景加入编译场景列表。



3）点击 Player Settings，将 Bundle Identifier 设置为 com.coolshow.videotest。



4）连接 Android 设备，将 Android 手机设为 USB 调试模式，点击 Build And Run 运行该程序。对准识别图，会播放测试视频。



2.4　制作 AR 对战游戏

2.4.1　制作识别图

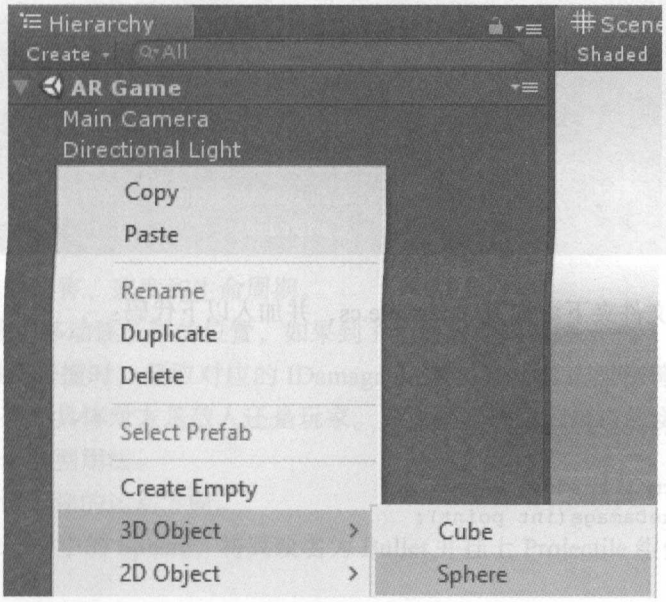
准备敌人（Enemy.jpg）和玩家角色（Player.jpg）的识别图，类似 2.3.1 节的做法，上传识别图。注意本次的 width 设为 1。

上传成功后，下载更新识别图数据库。

2.4.2　创建子弹

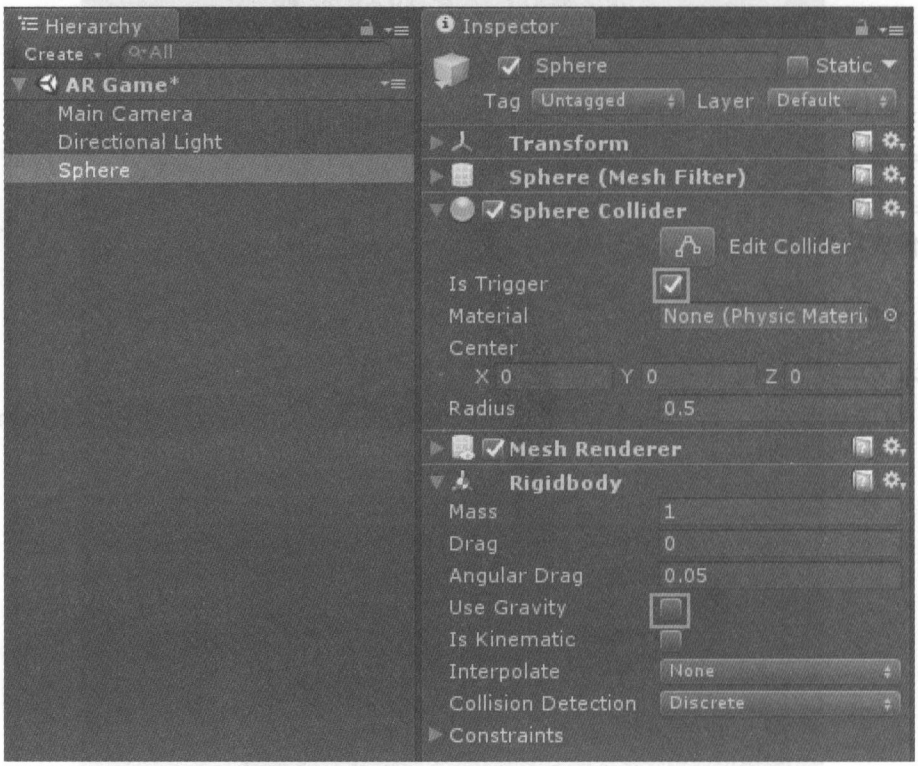
在本节中我们创建子弹的逻辑。

1）在场景中创建一个 Sphere：在 Hierarchy 面板中点击右键，依次选择 3D Object → Sphere。



2）将 Sphere 的 Scale 设为 0.1，0.1，0.1。

3）为 Sphere 添加 Rigidbody 组件，取消勾选 Use Gravity，并为 Sphere Collider 勾选 Is Trigger。



4）在 Script 文件夹下添加 IDamagable.cs，并加入以下代码：

```
using UnityEngine;

public interface IDamagable
{
    Transform transform { get; }
    void TakeDamage(int point);
}
```

在该接口中有一个 get 是可以获取一个 Transform，MonoBehavior 默认实现了这个 get 方法。定义 TakeDamage 方法的原因会在后面做详细说明。

5）在 Script 文件夹下添加 Projectile.cs，并加入以下代码：

```
using UnityEngine;

public class Projectile : MonoBehaviour
{
    public int   damage   = 1;
    public float speed    = 1;
    public float lifeTime = 5;

    float mLivedTime;
    Vector3 mMoveSpeed;

    void Update()
    {
        transform.Translate(mMoveSpeed * Time.deltaTime);
        mLivedTime += Time.deltaTime;
        if (mLivedTime >= lifeTime) {
            Destroy(gameObject);
        }
    }

    void OnTriggerEnter(Collider other)
    {
        IDamagable target = other.GetComponentInParent<IDamagable>();
        if (target != null) {
            target.TakeDamage(damage);
            Destroy(gameObject);
        }
    }

    public void Fire(Vector3 direction)
    {
        mLivedTime = 0;
        mMoveSpeed = direction.normalized;
        mMoveSpeed *= speed;
    }
}
```

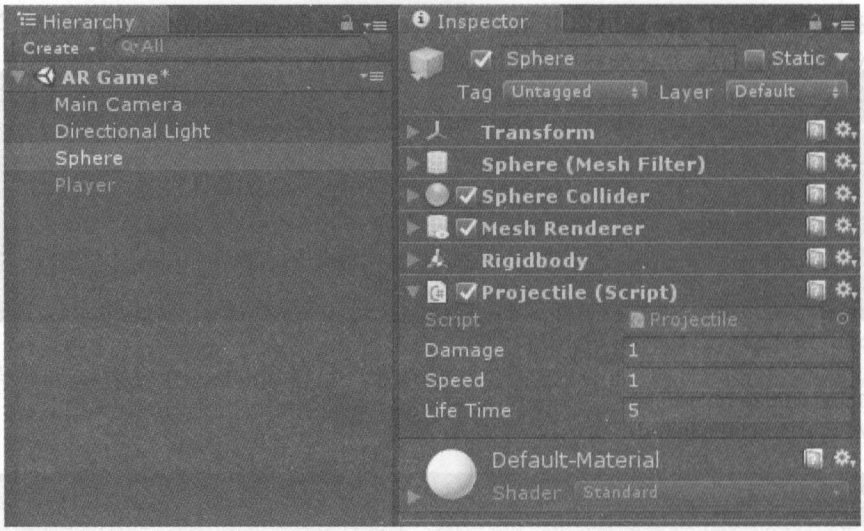
首先定义子弹的伤害、速度和生命周期。

在 Update 方法中移动该子弹的位置，如果到了生命周期则销毁。

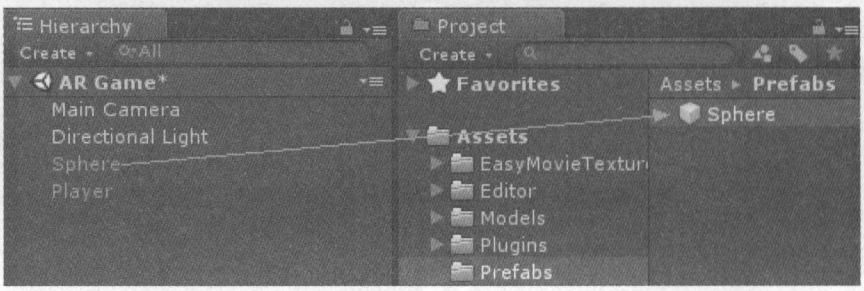
在碰撞器检测到碰撞时，获取对应的 IDamagable 接口并对其造成伤害。因为使用了该接口，因此不需要判断具体对方是敌人还是玩家。只要是实现了该接口的实例都可以受到伤害。这是多态的一个典型用法。

Fire 方法设定该子弹的运动方向。

6）选中 Hierarchy 中的 Sphere，将其改名为 Bullet 并挂上 Projectile 组件。



7）在 Assets 目录下创建 Prefab 文件夹，将其拖至该文件夹生成 Bullet 的 Prefab。



8）删除 Hierarchy 中的 Sphere，并将 Sphere 的 Prefab 改名为 Projectile。

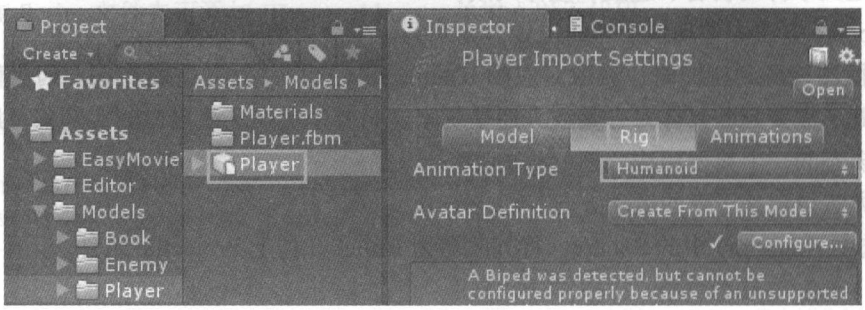
2.4.3 创建玩家角色

本节将创建一个可以近战攻击，远程攻击，并可以受伤的角色。

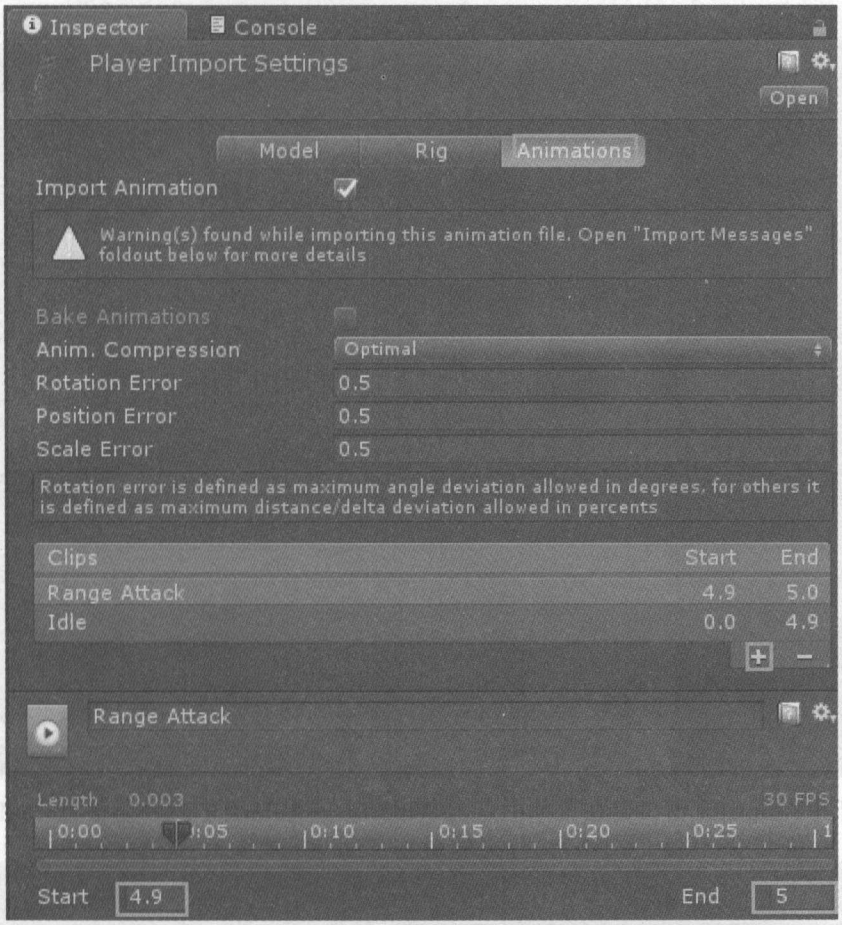
1）在 Assets/Scenes 目录下创建空场景 AR Game。

2）创建文件夹 Assets/Models/Player，将玩家的模型文件 Player.fbx 放在该文件夹下。

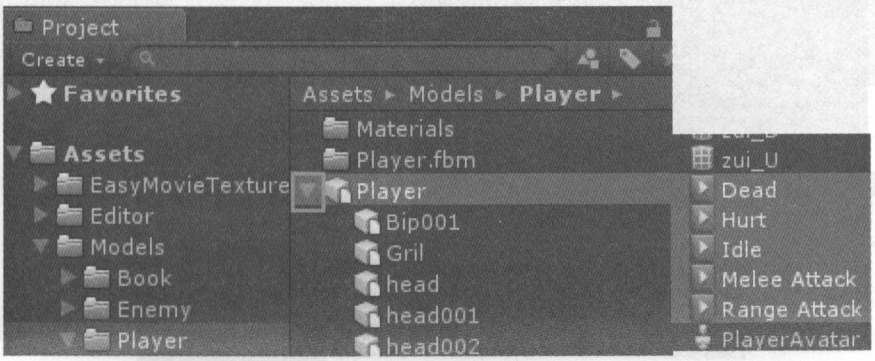
3）选中 Player，确保 Player 的动画为 Humanoid。



4）在 Animations 选项卡中根据“动作帧数 .txt”的说明按照帧数切割动画为 Idle，Range Attack，Melee Attack，Hurt 和 Dead。

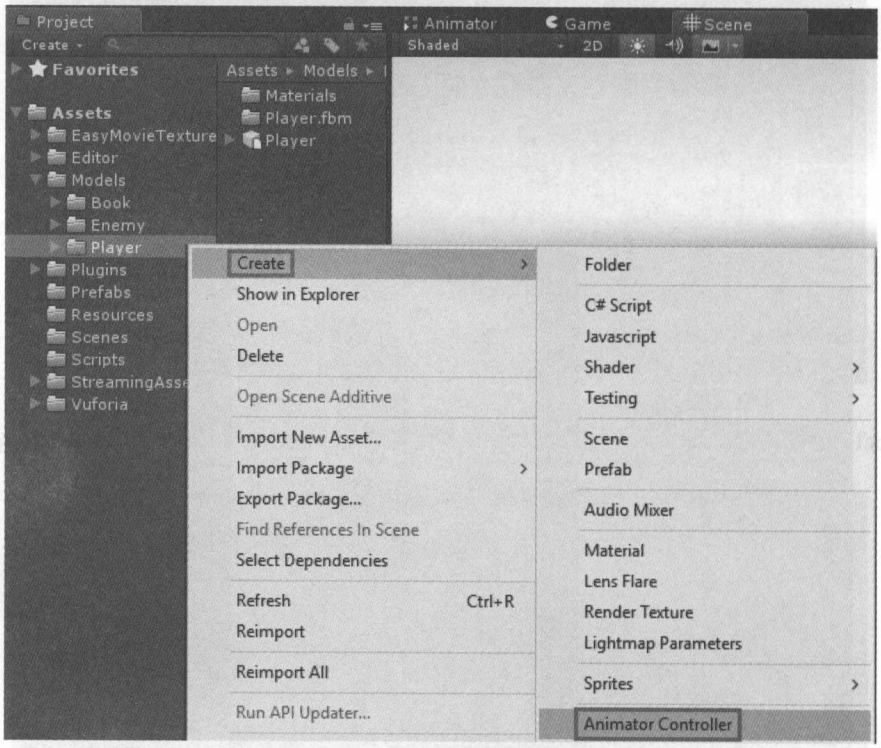


5）在 Project 面板中展开 Assets/Models/Player/Player，选中刚才创建的动画，按下组合键 Ctrl + D 复制这些动画到当前文件夹（因为模型中的动画是只读的，复制后才能添加 Animation Event 等写操作）。

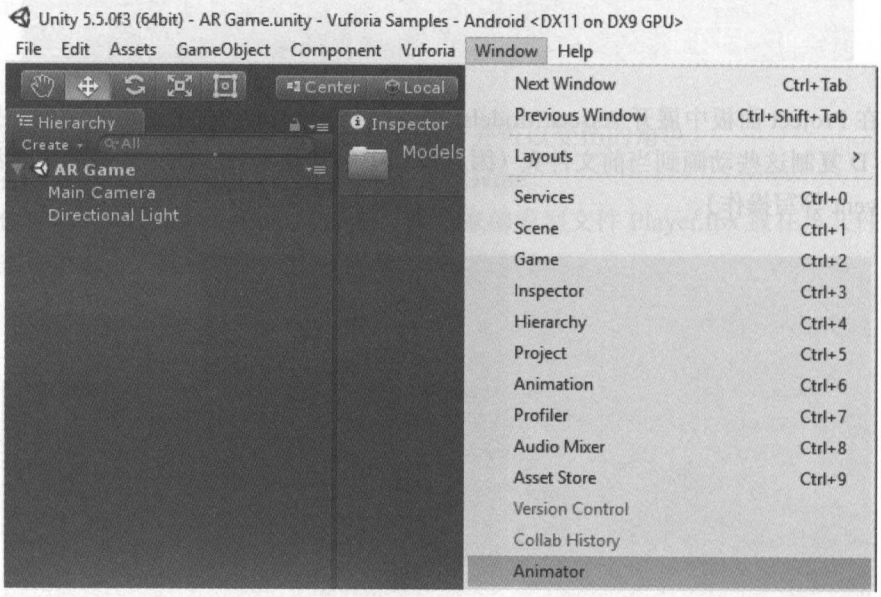


6）将 Player 拖至 Hierarchy 面板。

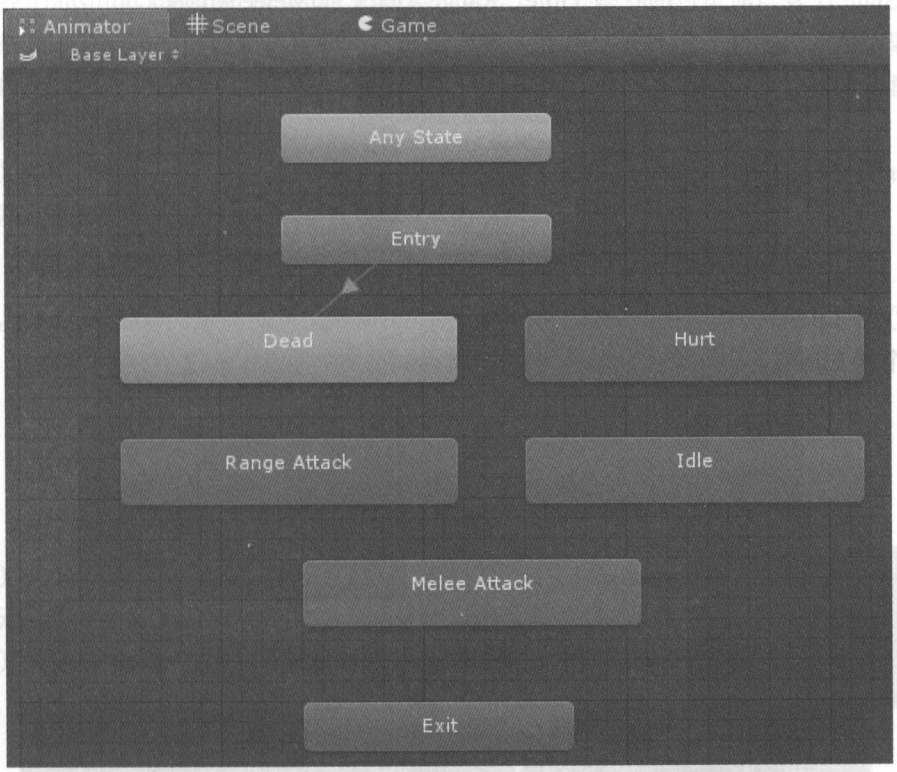
7）在 Player 文件夹上点击右键，依次选择 Create → Animator Controller 创建一个动画控制器。



8）打开 Animator 窗口，在菜单栏点击 Window → Animator。

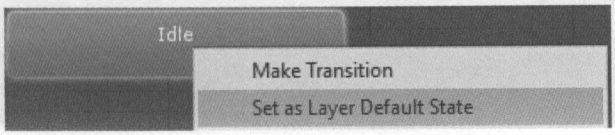


9）在 Hierarchy 面板中选中 Player，观察 Animator 窗口可以看到 Player 的动画状态机。

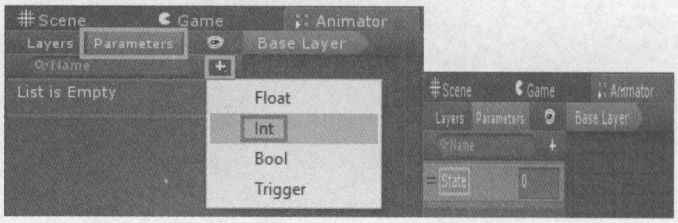


10）将每个状态的 Clip 替换为刚才复制出来的 Animation Clip（以 Idle 为例：选中 Idle，将右侧的 Motion 替换为刚才复制出来的 Idle 动画）。

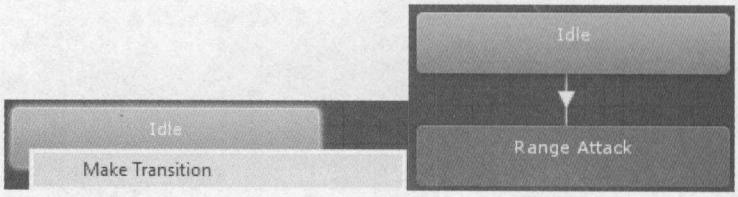
11）在 Idle 状态上点击右键，点击 Set as Layer Default State，将空闲状态设为状态机的默认状态。设置后，在游戏开始时，会播放玩家角色的默认动画：Idle（空闲）动画。



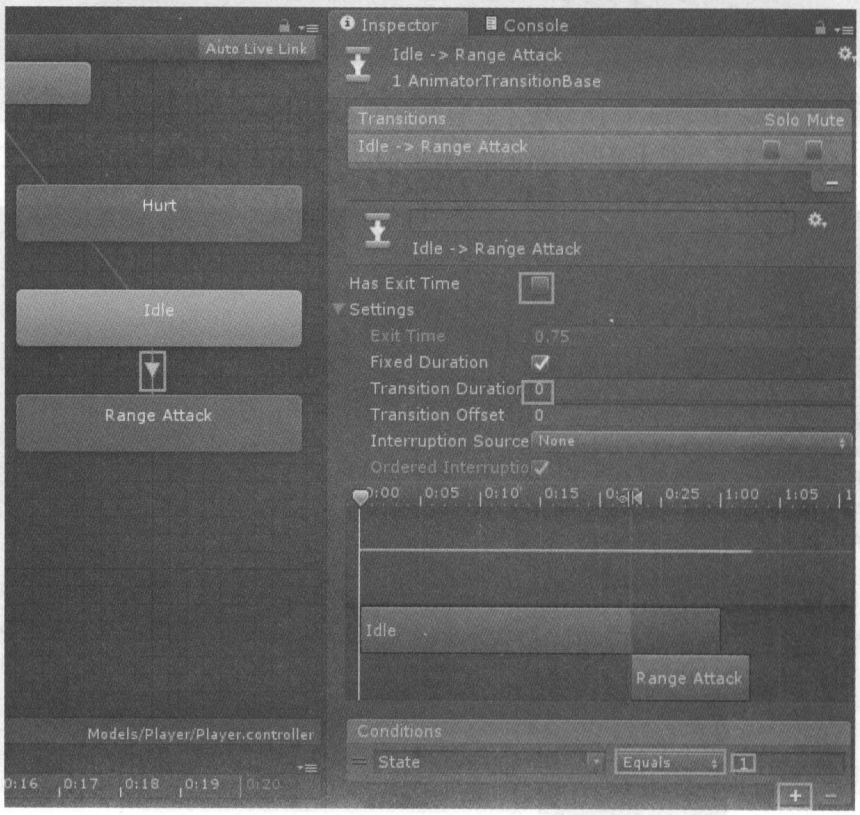
12）在 Animator 面板左侧切换至 Parameters 选项卡，点击“+”，选择 Int，添加一个整型变量 State 代表当前角色的状态（Idle、Range Attack 或 Melee Attack）。



13）在 Idle 状态上点击右键，点击 Make Transition，将箭头指向 Range Attack。意味着存在一个从 Idle 状态到 Range Attack 状态的转换。

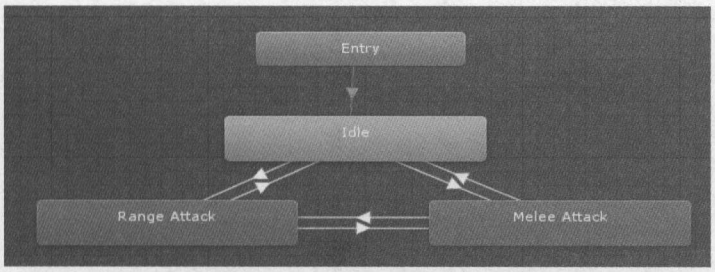


14）鼠标左键点击刚才创建的线，编辑状态转换的逻辑如下：



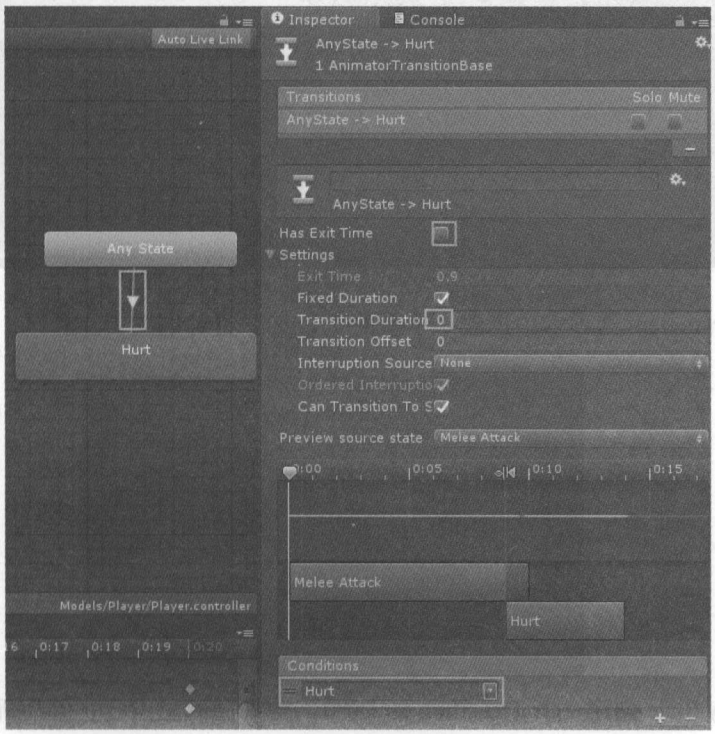
- 取消勾选 Has Exit Time，也就是不需要在 Idle 动画完成后再切换动画。
- 将 Transition Duration 设为 0，也就是不需要自动补帧动画。
- 在 Inspector 面板中设置转换条件：点击“+”，在条件处选择“Equal”，在结果处输入 1，也就是当 State=1 的时候，角色从 Idle 状态转换成 Range Attack 状态（即动画播放的转换）。

15）类似上面的方式，当 State=2 时，使动画切换到 Melee Attack 状态。当 State=0 时，使动画切换到 Idle 状态。完成后的状态图如下图所示。

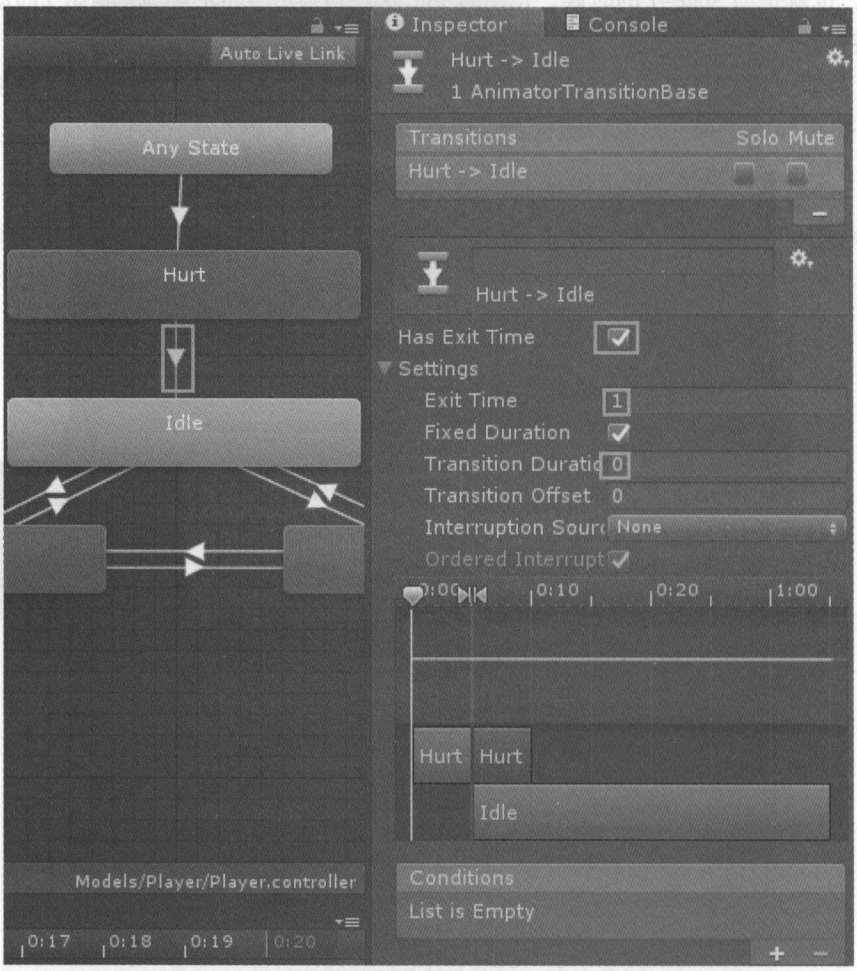


16）在 Parameter 选项卡处添加一个名为 Hurt 的 Trigger 来表示玩家是否受伤了。

17）从 Any State 创建一个 Transition 到 Hurt，类似 State 的设定，但将判断条件设为 Hurt，也就是无论当前角色处于什么状态，如果受伤了则直接终止当前状态进入受伤状态，并播放受伤的动画。

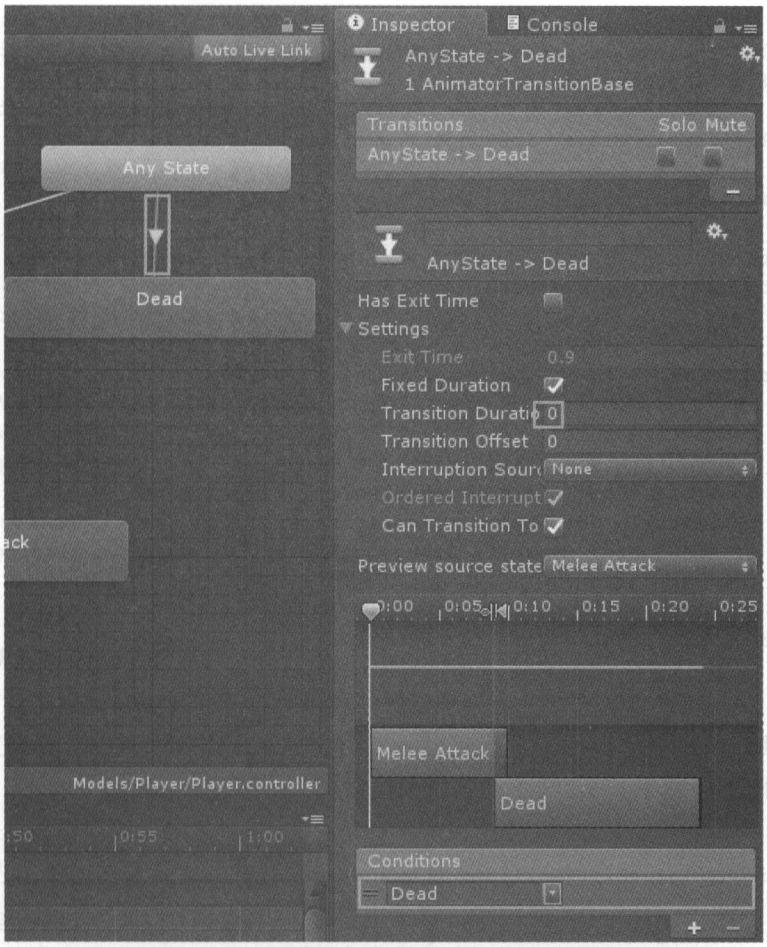


18）当受伤状态结束后，默认进入Idle状态。因此创建一个从Hurt到Idle的Transition并把Has Exit Time的时间设为1。也就是说，这个状态的动画结束后再进入Idle状态。



19）在Parameters选项卡中创建一个新的Trigger，并命名为Dead。

20）从Any State创建一个Transition到Dead，类似Hurt的设定，但判断条件设为Dead。



21）在 Script 文件夹下添加 Player.cs，并加入以下代码：

```
using UnityEngine;
using System.Collections.Generic;

public class Player : MonoBehaviour, IDamagable
{
    public enum State
    {
        Idle,
        RangeAttack,
        MeleeAttack
    }

    const string ANIM_PARAM_STATE = "State";
    const string ANIM_PARAM_HURT  = "Hurt";
    const string ANIM_PARAM_DEAD  = "Dead";

    public int damage = 2;
    public int health = 10;
```

```
    public Projectile projectilePrefab;
    public Transform  fireSpot;

    public static Player instance { get; private set; }

    Animator mAnimator;

    State mCurrentState;
    List<IDamagable> mTargets;


    void Start()
    {
        instance = this;
        mCurrentState = State.Idle;
        mTargets = new List<IDamagable>();
        mAnimator = GetComponent<Animator>();
        if (mAnimator == null) Debug.LogError( "Animator is missing!" );
    }

    void OnTriggerEnter(Collider other)
    {
        var target = other.GetComponentInParent<IDamagable>();
        if (target != null) {
            mTargets.Add(target);
        }
    }

    void OnTriggerExit(Collider other)
    {
        var target = other.GetComponentInParent<IDamagable>();
        if (target != null) {
            mTargets.Remove(target);
        }
    }

    public void UpdateState(int state)
    {
        mCurrentState = (State)state;
        mAnimator.SetInteger(ANIM_PARAM_STATE, (int)mCurrentState);
}

    public void TakeDamage(int point)
    {
        health -= point;
        if (health > 0) {
            mAnimator.SetTrigger(ANIM_PARAM_HURT);
        } else {
            mAnimator.SetTrigger(ANIM_PARAM_DEAD);
        }
    }
```

```
    #region Animation Callback
    public void Fire()
    {
        if (mTargets.Count == 0) return;

        foreach (var t in mTargets) {
            t.TakeDamage(damage);
        }
    }

    public void FireProjectile()
    {
        var proj = Instantiate(projectilePrefab, fireSpot.position, Quaternion.identity);
        Vector3 dir = fireSpot.position - transform.position;
        dir.y = 0;
        proj.Fire(dir);
    }
    #endregion
}
```

首先定义玩家的三个状态 Idle、Range Attack 和 Melee Attack，对应的整数分别是 0，1，2。对应动画状态机的三个数字。

然后在常量中定义状态机的三个参数名。

接下来定义角色的近战攻击力和生命值，远程攻击时使用的子弹的 Prefab 和子弹的发射点的引用，instance 是 Player 的实例（因为全局只有一个 Player（单例））使用 mTargets 列表获取已经进入攻击范围的敌人列表。

在 Start 方法中初始化状态，并获取 Animator 组件。

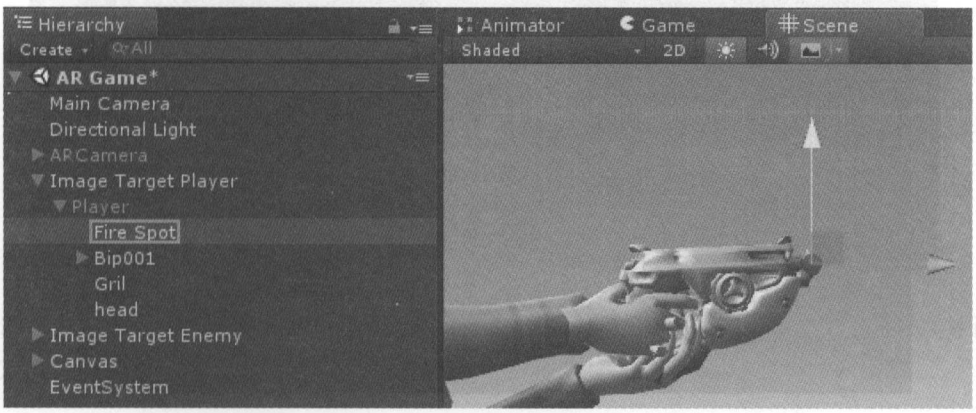
在 OnTriggerEnter 和 OnTriggerExit 事件中更新目标列表。

提供一个公有方法 UpdateState 来更新角色的状态。

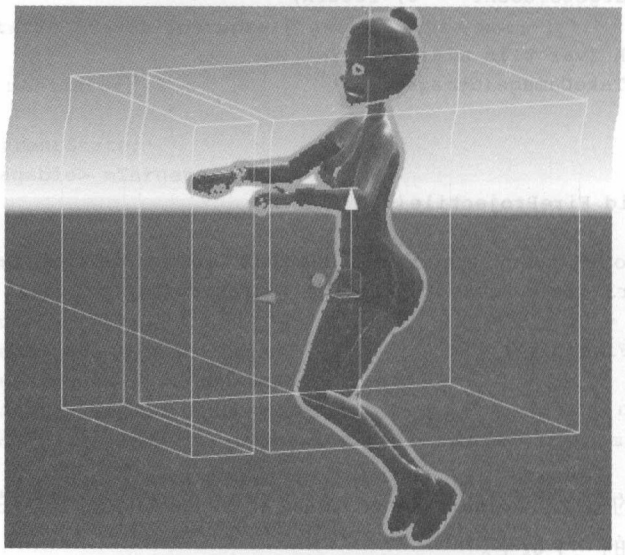
提供一个公有方法 TakeDamage 来承受伤害，当角色 health 为 0 时切换到死亡动画。

Fire 和 FireProjectile 方法是动画的回调方法，由 Animation Event 调用。

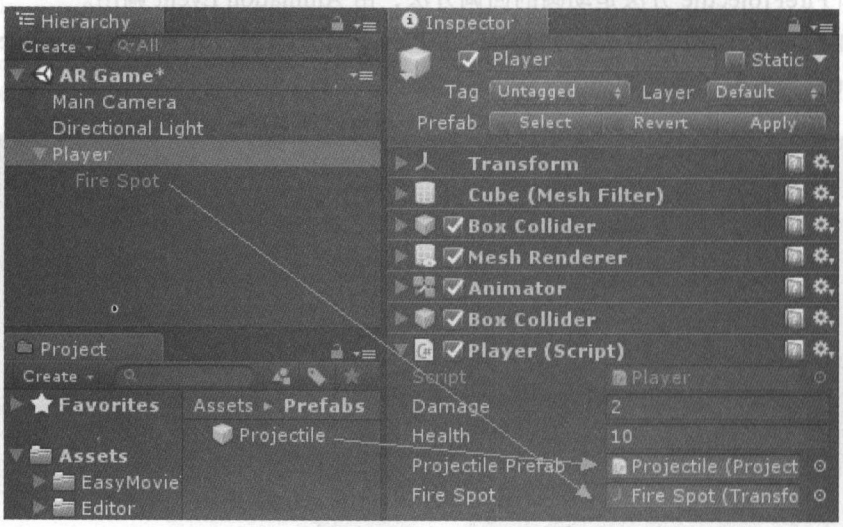
22）在 Hierarchy 面板中选中 Player，在 Player 前方添加一个空的 GameObject 并命名为 Fire Spot 作为发射子弹的起点。



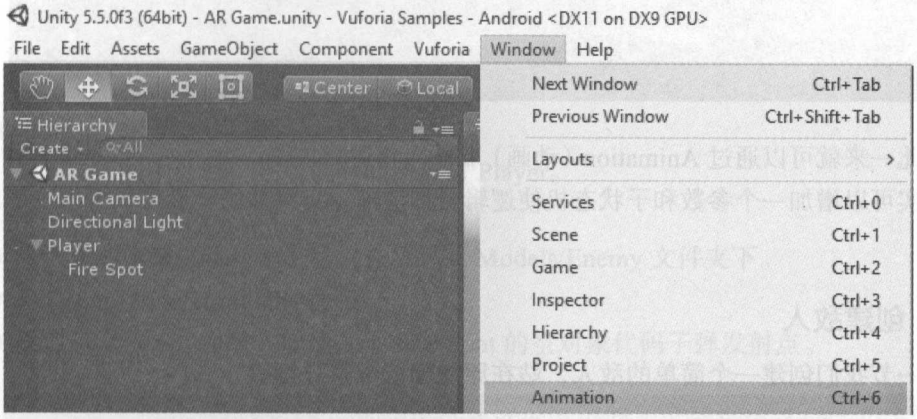
23）为 Player 添加两个 BoxCollider 并设为 Trigger，一个放置在 Player 身上作为受击目标，另一个作为攻击范围的检测区域。



24）添加 Player 脚本，并将 Bullet 和 Fire Spot 拖至对应位置。



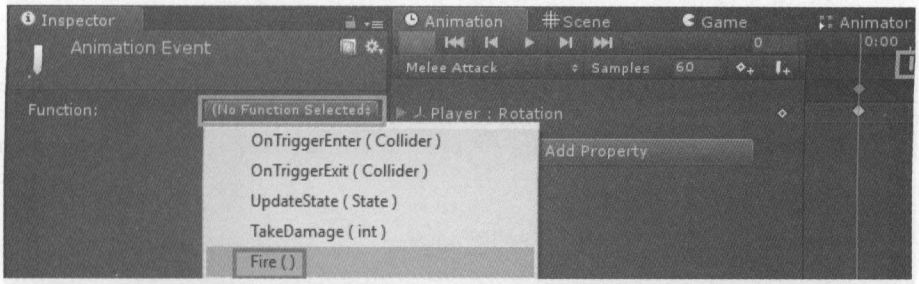
25）打开 Animation 窗口，在菜单栏中点击 Window → Animation。



26）在 Hierarchy 面板中选中 Player，选择编辑动画为 Melee Attack（近战攻击）。

27）拖动红色的竖线选择指定动画帧处，点击图中所示按钮，在合适的帧处添加一个 Animation Event。

28）点击这个白色的竖线选择该动画事件，在 Inspector 面板中选择回调方法为 Fire。



29）使用类似上面的方法，为 Range Attack 动画也添加一个 Animation Event，并设置回调方法为 FireProjectile( )。

如此一来就可以通过 Animation（动画）驱动人物进行攻击。

其实可以增加一个参数和子状态机使逻辑更加清晰。有兴趣的读者可以考虑一下应该如果实现。

2.4.4 创建敌人

这一节我们创建一个简单的敌人，站在原地朝着玩家开枪。

1）在 Assets/Scripts 目录下创建 Enemy.cs，并添加以下代码：

```
using UnityEngine;

public class Enemy : MonoBehaviour, IDamagable
{
    public int health = 2;
    public float attackCycle = 3;

    public Transform  fireSpot;
    public Projectile projectilePrefab;

    float timeCounter;


    void Update()
    {
        timeCounter += Time.deltaTime;
        if (timeCounter >= attackCycle) {
            timeCounter = 0;
            Fire();
        }
    }

    void Fire()
    {
          Projectile bullet = Instantiate(projectilePrefab, fireSpot.position,
Quaternion.identity);
        bullet.Fire(Player.instance.transform.position - fireSpot.position);
    }

    public void TakeDamage(int point)
    {
        health -= point;
        if (health <= 0) {
            Destroy(gameObject);
        }
```

```
    }
}
```

开头类似于 Player，是敌人的血量、发射点和子弹的 Prefab。这里的 attackCycle 代表攻击周期，这里默认敌人每 3 秒发射一次子弹攻击玩家。下面的 timeCounter 就是计算这个周期使用的变量。

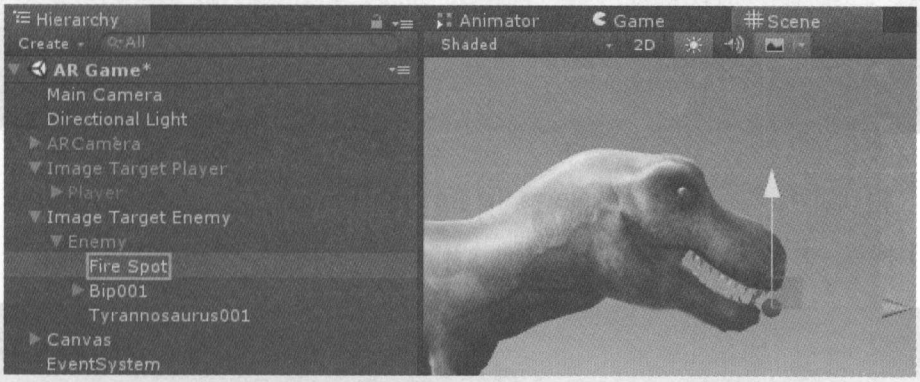
在 Update 方法中计时，到周期时间后攻击 Player。

在 TakeDamage 方法中进行伤害处理。

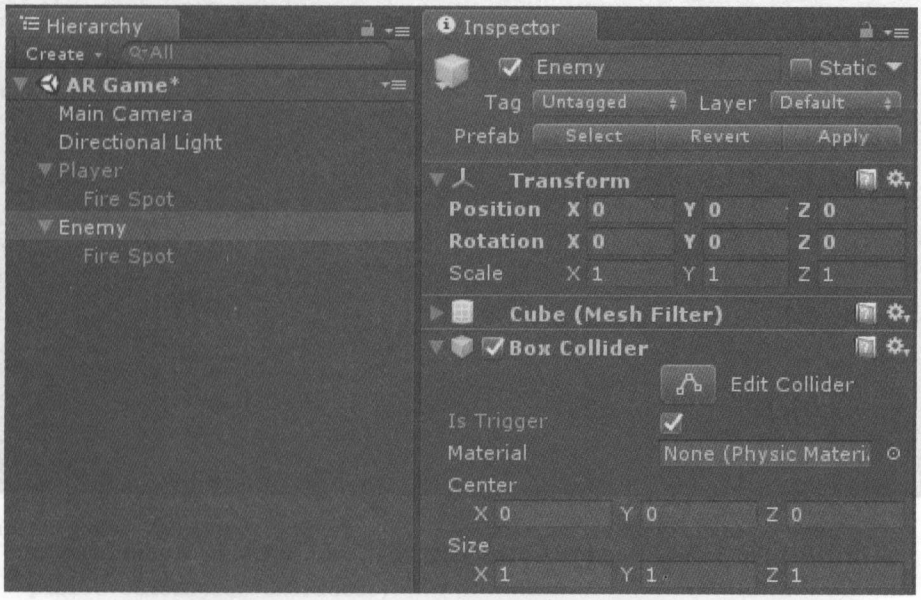
2）将敌人的模型 Enemy.fbx 存放在 Assets/Models/Enemy 文件夹下。

3）将 Enemy 拖至 Hierarchy 面板。

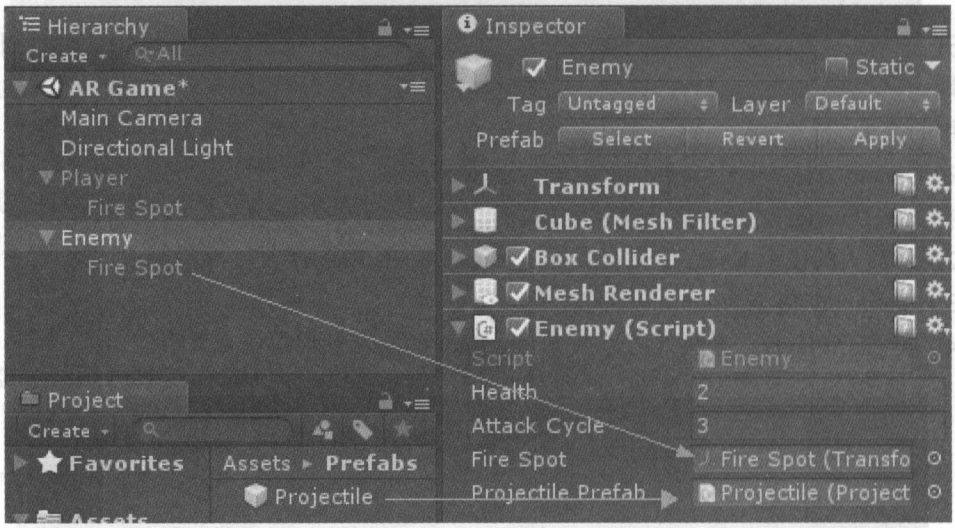
4）类似于 Player，为敌人创建一个 Fire Spot 的空对象代码子弹发射点。



5）为 Enemy 添加 Box Collider 并勾选 Is Trigger。

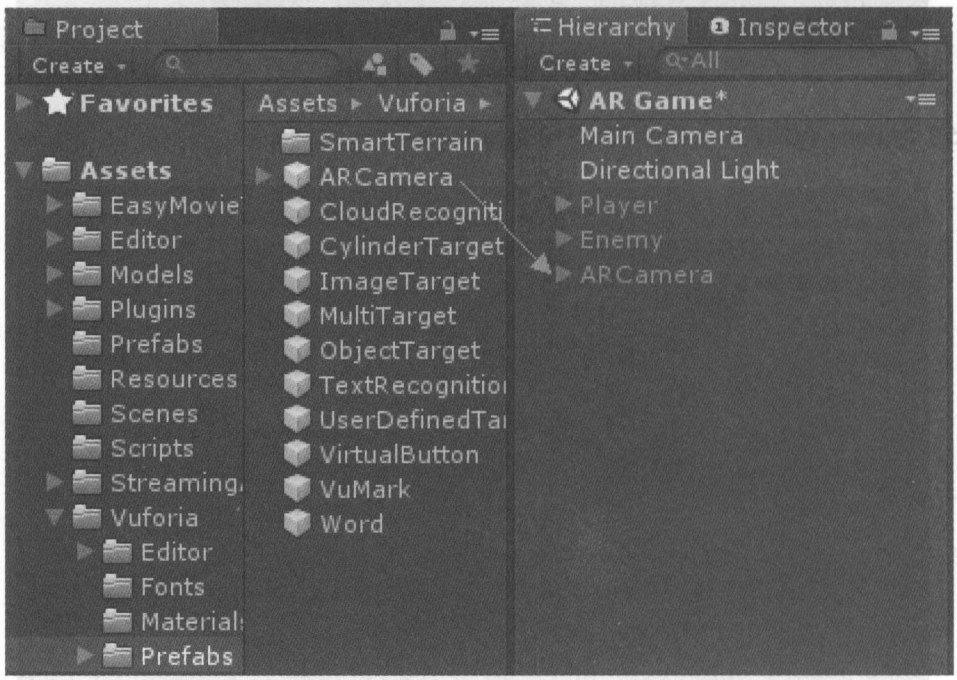


6）为 Enemy 添加 Enemy.cs 脚本，添加 Fire Spot 和 Projectile 的引用。

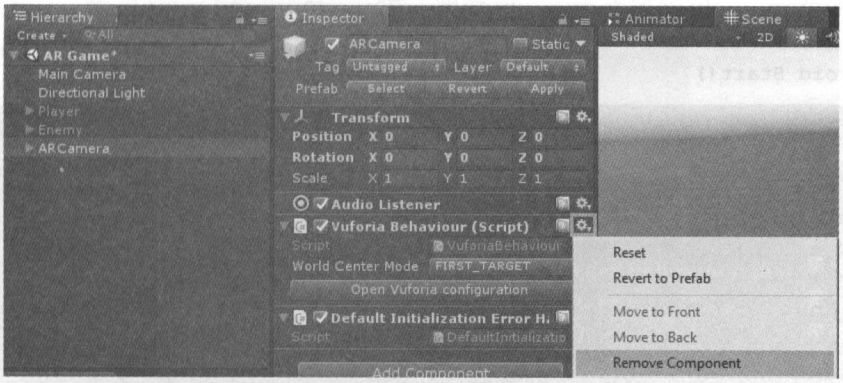


2.4.5 将玩家和角色设定为 Image Target

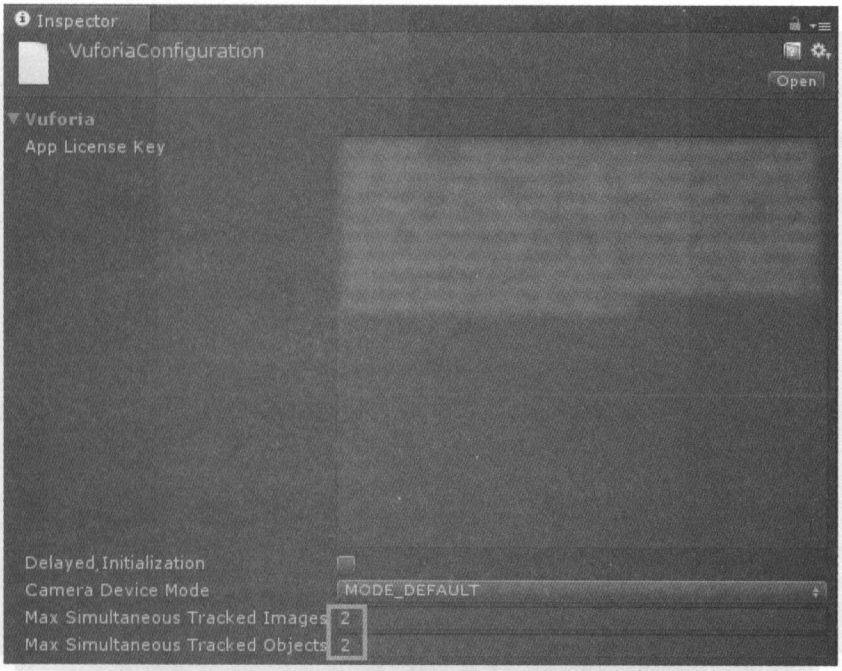
1）将 Assets/Vuforia/Prefabs 目录下的 ARCamera 拖至 Hierarchy 面板。



2）点击图中齿轮，选择 Remove Component 删除 ARCamera 的 Audio Listener。



3）在VuforiaConfiguration中确认Study数据库已被加载并激活，设置Max Simultaneous Tracked Images和Max Simultaneous Tracked Objects为2，意味着最多同时出现两个识别图和识别物体（分别是敌人和玩家角色）。



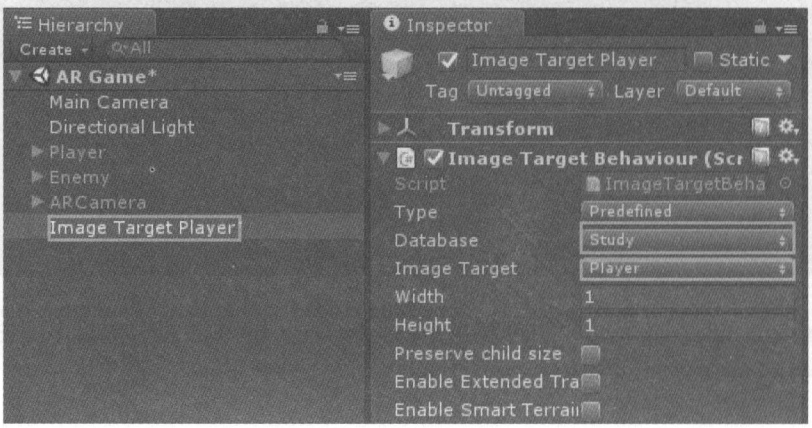
4）在Assets/Scripts目录下创建CharacterTracker.cs，并添加以下代码：

```
using UnityEngine;
using Vuforia;


public class CharacterTracker : MonoBehaviour, ITrackableEventHandler
{
    public GameObject character;
```

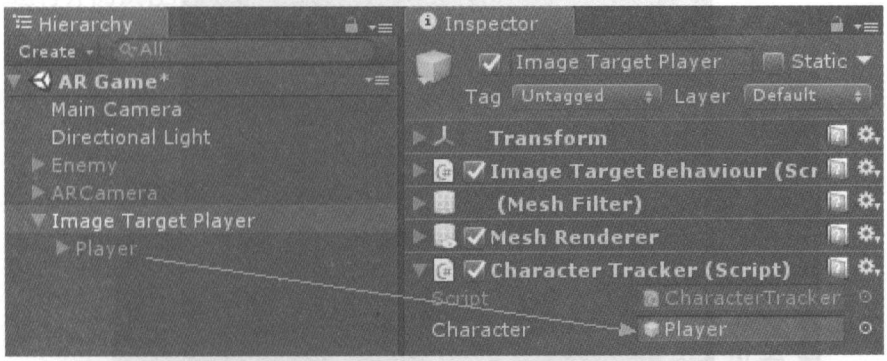
```
    TrackableBehaviour mTrackableBehaviour;

    void Start()
    {
        mTrackableBehaviour = GetComponent<TrackableBehaviour>();
        if (mTrackableBehaviour) {
            mTrackableBehaviour.RegisterTrackableEventHandler(this);
        }
    }

    public void OnTrackableStateChanged(TrackableBehaviour.Status
previousStatus, TrackableBehaviour.Status newStatus)
    {
        if (newStatus == TrackableBehaviour.Status.DETECTED ||
            newStatus == TrackableBehaviour.Status.TRACKED ||
            newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED) {
            OnTrackingFound();
        } else {
            OnTrackingLost();
        }
    }

    void OnTrackingFound()
    {
        character.SetActive(true);
    }

    void OnTrackingLost()
    {
        character.SetActive(false);
    }
}
```

整体功能类似于 VideoTracker.cs，在响应事件中实现基本的打开 / 关闭 GameObject。

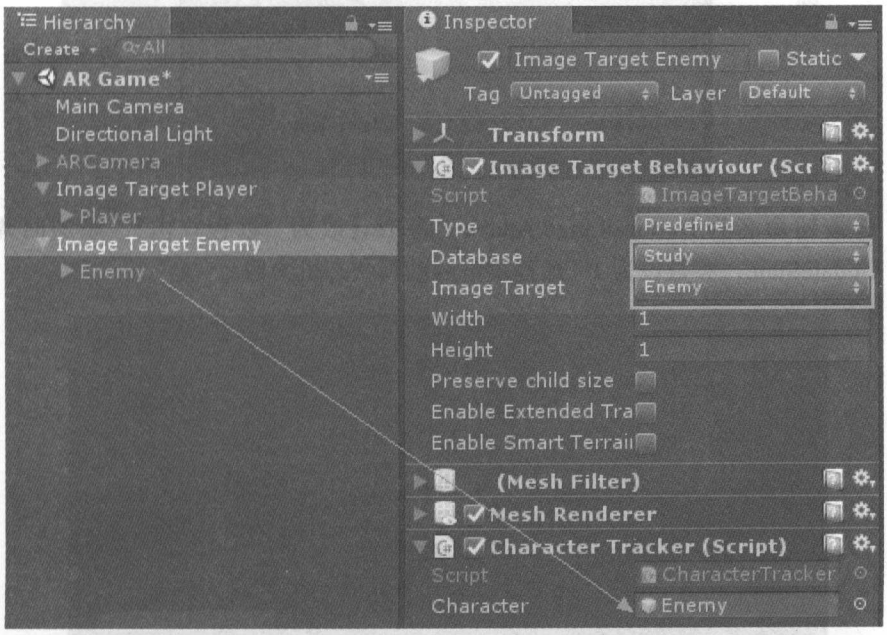
5）在 Hierarchy 中创建一个空的 GameObject，命名为 Image Target Player，为其添加 Image Target Behaviour 和 Character Tracker 组件，并设置 Database 为 Study，Image Target 为 Player。



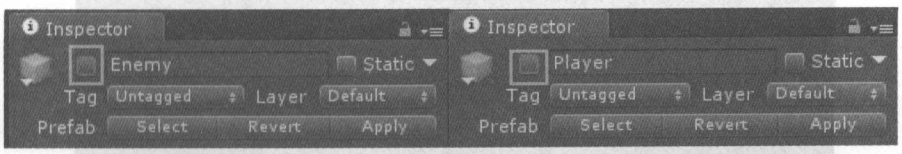
6）将 Player 拖至 Image Target Player 下，调整 Player 的位置使 Player 站在识别图的上方。并将 Player 的引用拖至 Character Tracker 的 Character 下。



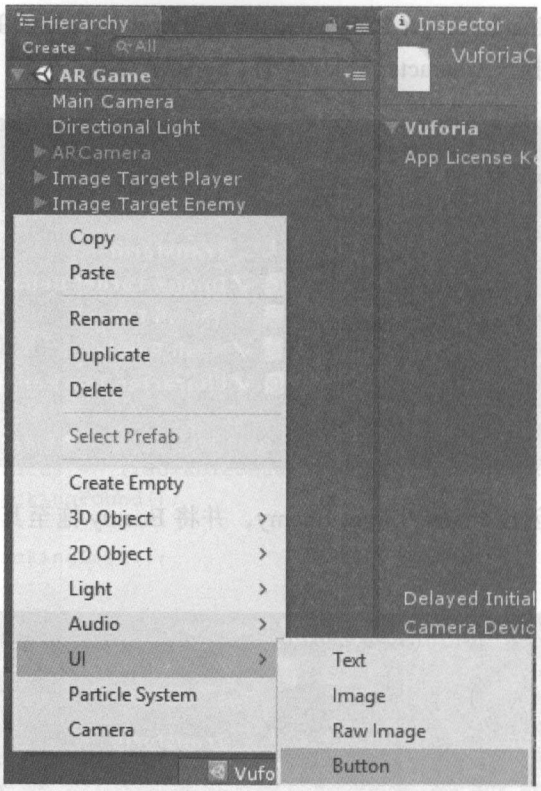
7）类似上面两步创建 Image Target Enemy，并将 Enemy 拖至其 Transform 下方，并更新 Character 的引用。



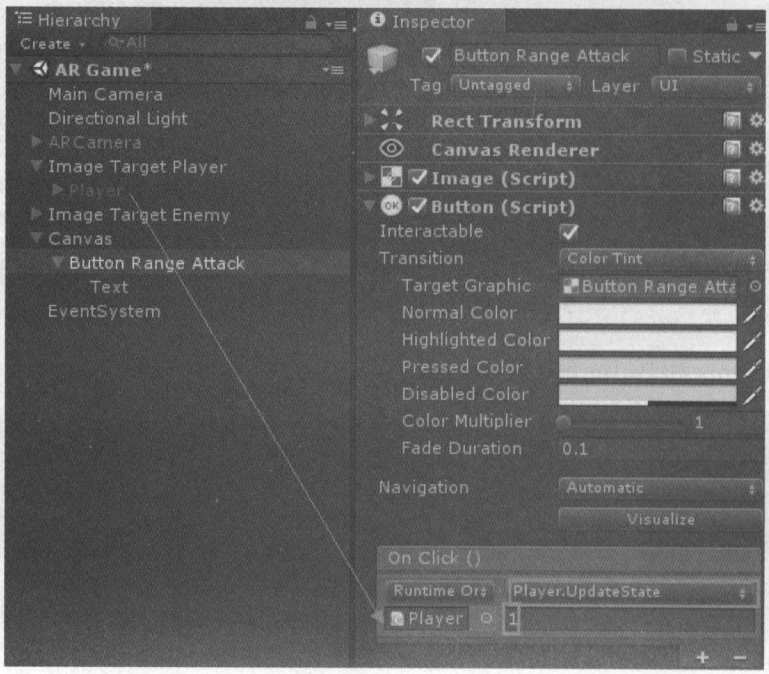
8）关闭 Enemy 和 Player 的 GameObject。



9）创建一个按钮。在 Hierarchy 中点击右键，依次选择 UI → Button，并将按钮放置在右下角。

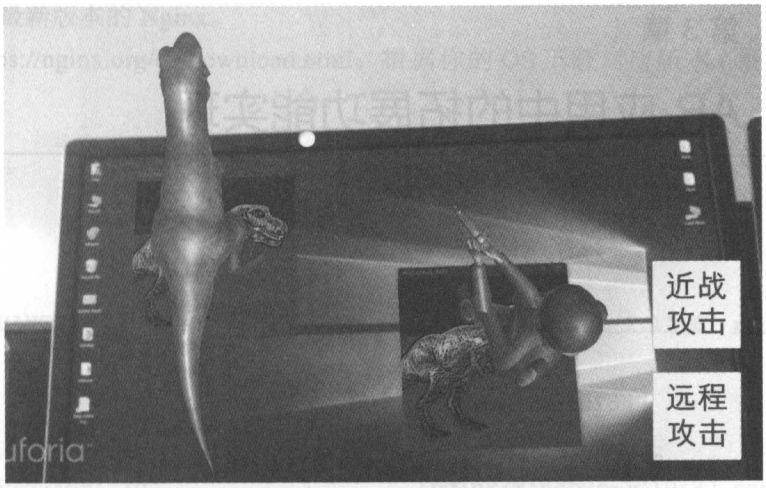


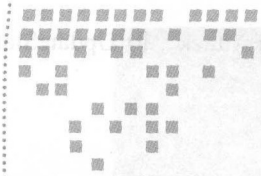
10）将按钮命名为 Button Range Attack，将 Text 改为“远程攻击”，添加回调函数为 Player 的 UpdateState，参数设为 1（远程攻击的动画状态机参数）。



11）类似于 Button Range Attack，创建 Button Melee Attack，但 UpdateState 的参数为 2（近战攻击的动画状态机参数）。

12）运行并测试程序。





Chapter 3 第 3 章

AR 应用中的拓展功能实现

3.1 在 Unity 3D 中实现网络通信

在创建 Unity 应用时往往需要使用网络通信来获取一些实时数据，例如账号登录、内容更新、数据同步等。本章重点介绍一下 Unity3D 中的网络通信部分。

3.1.1 计算机网络简介

计算机网络 OSI 模型分为 7 层：

- 物理层
- 数据链路层
- 网络层
- 传输层
- 会话层
- 表示层
- 应用层

在 Unity 网络编程中，绝大部分的场景只需要考虑应用层的内容，如果需要自定义通信协议（例如网络游戏中的通信）则需要使用和设计传输层到应用层的内容（这些属于底层网络编程的内容，本书不做介绍）而物理层到网络层的内容偏硬件，实际应用中几乎不会遇到。

应用层有很多流行的协议，本章重点介绍如何使用 HTTP 和 HTTPS 协议从网络上获取数据。

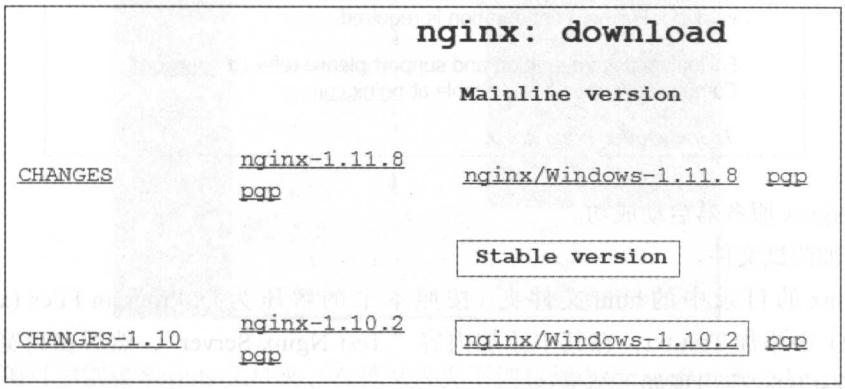
HTTP 即超文本传输协议（HyperText Transfer Protocol），是互联网上最流行的访问协议。虽然大部分常见的使用场景是获取并显示网页上的内容，但实际上可以用于各种一次性通信。底层其实就是一次服务器与客户的 TCP 通信。

3.1.2 搭建一个 HTTP 服务器

目前主流的 HTTP 服务器有很多，包括 IIS、Apache、Nginx 等。其中 Apache 和 Nginx 是免费开源的。而且性能高效且功能强大，这里我们选择 Nginx 作为我们的 HTTP 服务器。

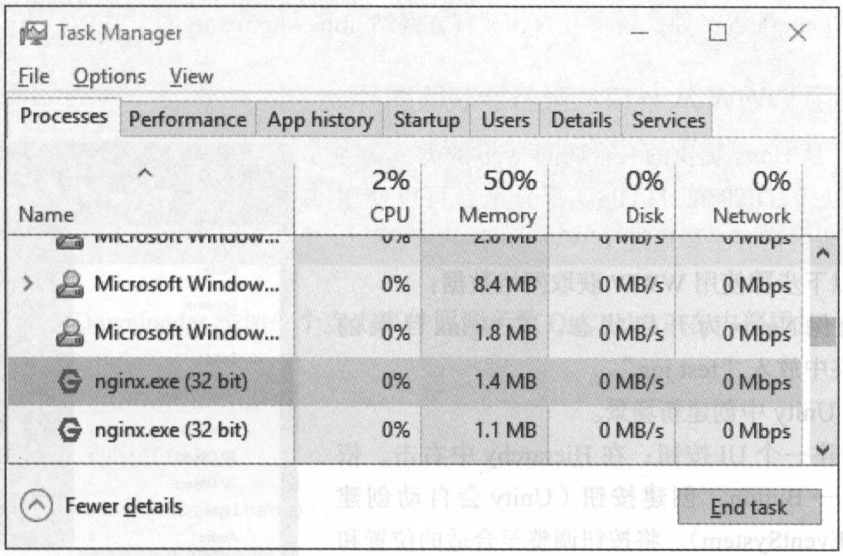
1）下载最新版本的 Nginx。

打开 https://nginx.org/en/download.html，根据你的 OS 下载对应版本（最好选择下载稳定版）。



2）运行 Nginx 服务器。

下载完毕后，将压缩包解压到任意目录。本书中为“D:\Program Files (x86)”目录下，双击 nginx.exe 即可启动服务器（由于目前的 Nginx 版本没有图形界面，因此默认启动命令行来启动 Nginx）可以在任务管理器中查看进程状态。



Nginx 默认会启动两个进程，一个管理进程和一个 HTTP 服务进程，结束时如果结束的是 HTTP 服务进程，管理进程会再生成一个服务进程。所以需要先结束管理进程再结束服务

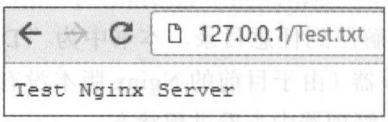
进程。通常管理进程比服务进程内存使用量低一些。

打开浏览器，输入 127.0.0.1，应该显示如下结果：

至此 Nginx 服务器启动成功。

3）添加测试文件。

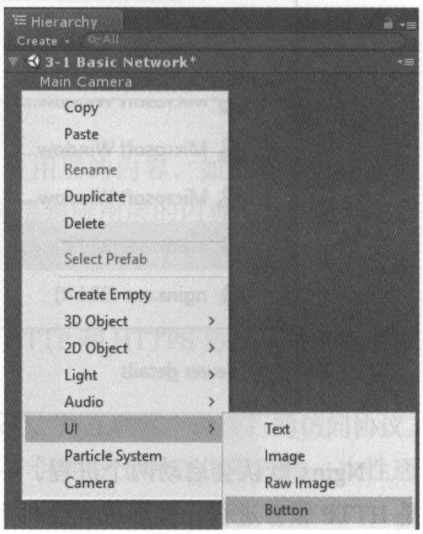
在 Nginx 的目录中的 html 文件夹（按照本书的解压为 D:\Program Files (x86)\nginx-1.10.2\html）中添加 Test.txt，在其中添加内容“Test Nginx Server”，然后在浏览器中输入 127.0.0.1/Test.txt，应出现如下结果：



之后本书中需要使用的素材也会放在 html 文件夹中，这是 Nginx 配置文件的默认目录，如何配置 Nginx 配置文件，请参考 Nginx 官方网站 https://nginx.org。

3.1.3 使用 WWW 从 HTTP 服务器获取图片

WWW 是 Unity 提供的一个访问 Web 的类，支持多种协议包括 HTTP 和 HTTPS。部分平台可以使用 FTP。接下来的内容主要通过 WWW 访问网络数据。



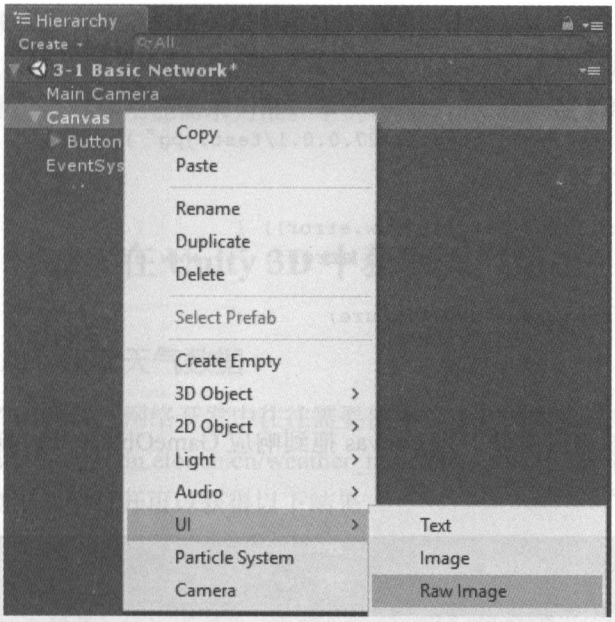
按照以下步骤使用 WWW 获取网络数据：

1）确保 Nginx 为开启状态，在 Nginx 目录的 html 文件夹中放入“test.jpg”。

2）在 Unity 中创建新场景。

3）创建一个 UI 按钮：在 Hierarchy 中右击，依次选择 UI → Button，创建按钮（Unity 会自动创建 Canvas 和 EventSystem）。将按钮调整至合适的位置和大小，并将 Button 的 Text 改为“Download”。

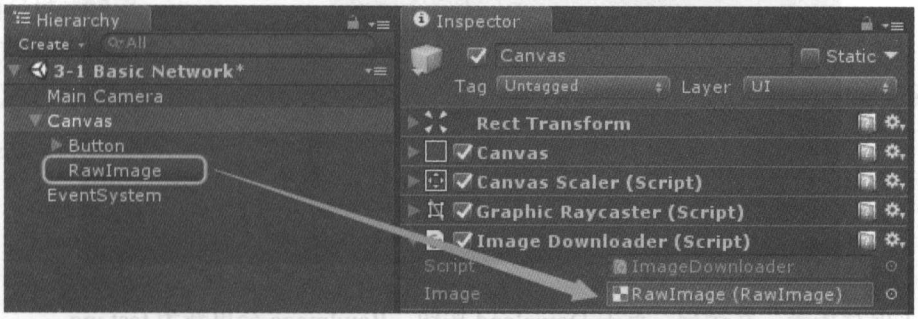
4）创建一个 Raw Image：在 Canvas 上点击右键，依次选择 UI → Raw Image。



5）在项目中创建 Scripts 文件夹，在该文件夹下创建脚本 ImageDownloader.cs。

6）添加一个公有成员 Image（类型为 RawImage）。

7）为 Canvas 添加 ImageDownloader 组件，将刚才创建的 RawImage 的引用赋予 Image-Dowloader.image。



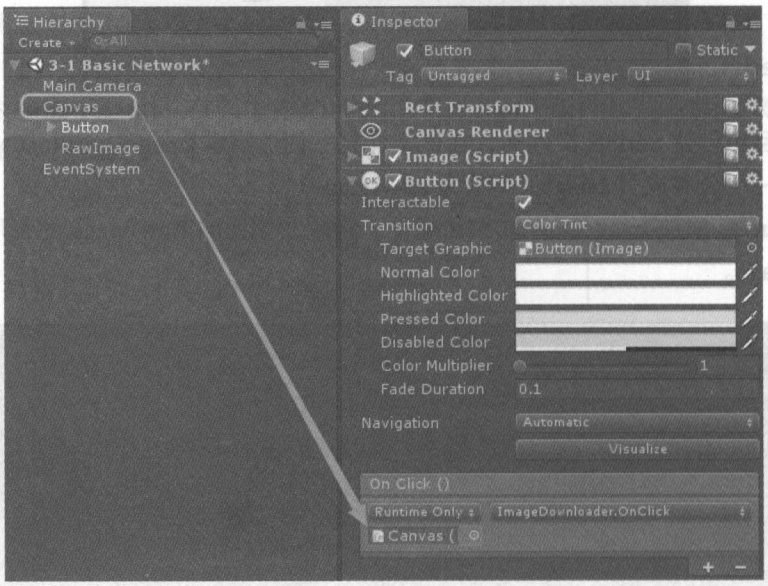
8）为 ImageDownloader 添加一个点击事件回调函数 OnClick()，在内部添加一个启动协程 Download。

```
public void OnClick()
{
    StartCoroutine(Download());
}
```

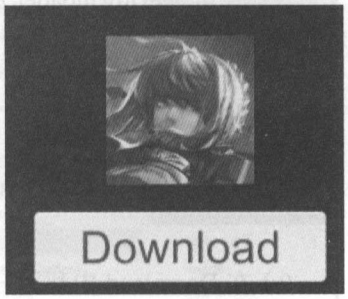
9）在 Download 中利用 HTTP 地址创建一个 WWW 对象 www，等待返回后将下载的图片（Texture2D）赋予刚才创建的 RawImage。

```
IEnumerator Download()
{
    // Get image from http server
    WWW www = new WWW( "http://127.0.0.1/test.jpg" );
    yield return www;

    if (!string.IsNullOrEmpty(www.error)) {
        Debug.LogError( "Download error : " + www.error);
    } else {
        image.texture = www.texture;
    }
}
```

10）为按钮添加响应事件，将 Canvas 拖到响应 GameObject 处，响应函数选择 ImageDownloader.OnClick。



11）保存场景，并运行。点击 Download 按钮，RawImage 会更新为 test.jpg。



在这个例子程序中我们通过 WWW 类访问自己搭建的 HTTP 服务器完成了一次 HTTP 请求，简单地讲一下网络通信中的原理。

1）WWW 的对象进行 HTTP 请求，等待结果。

2）服务器（Nginx）接收到请求，处理本次请求。本次的请求是一个简单的文件下载，实际运用中也可以有很多其他逻辑处理操作，然后返回所需数据。

3）服务器（Nginx）向客户端（Unity 应用）传输数据。

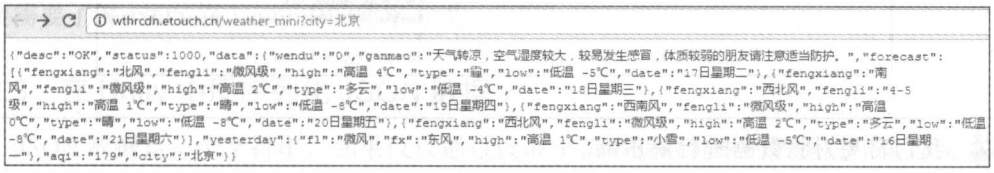
此次请求即图片的数据，由于是本机的回环地址所以速度非常快。

4）当传输完成，服务器（Nginx）断开本次链接（这其实是 HTTP 协议的一部分），这后续的操作就很自由了，以具体功能需求为准。本例中是 Unity 收到数据后将下载好的图片显示到 RawImage 上。

3.2　在 Unity 3D 中获取天气信息

3.2.1　通过网络 API 获取天气数据

在这个数据爆炸的时代，网络开发中往往需要使用第三方的数据接口。本例中使用的访问接口的格式为：http://wthrcdn.etouch.cn/weather_mini?city= 北京。其中的城市名可以换成任意的城市名，在浏览器中打开可以获得以下结果：



浏览器可以获取的数据用 Unity 中的 WWW 同样可以获得，这些网络 API 可以获取的数据都可以在 Unity 中很便利地获取，这为我们的应用开发提供了极大的便利。

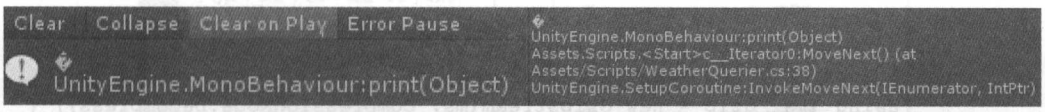
项目搭建步骤：

1）新建一个场景，创建 WeatherQuerier.cs 脚本，拖至 Main Camera 上。

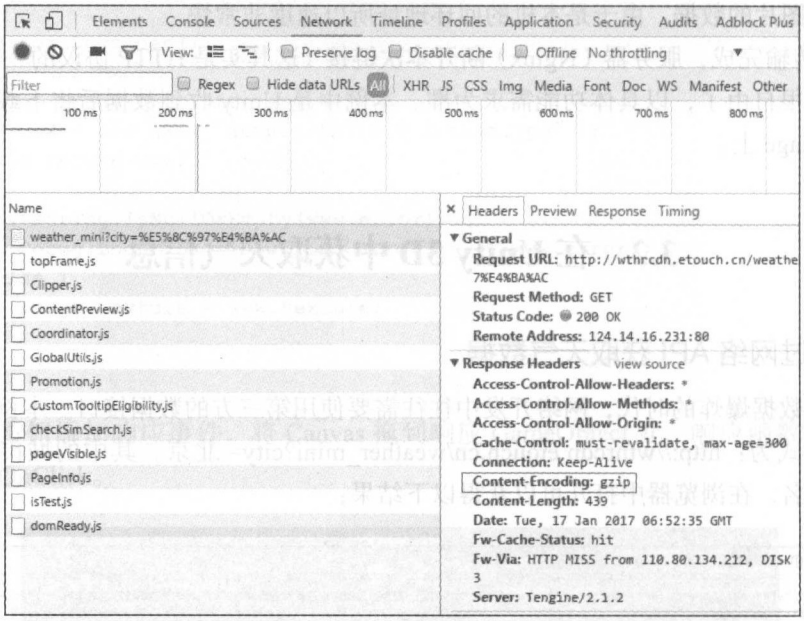
2）在 Start 中通过上述网络 API 获取天气数据，并将 Start 的返回值改为 IEnumerator。

```
IEnumerator Start()
{
    WWW www = new WWW("http://wthrcdn.etouch.cn/weather_mini?city=北京");
    yield return www;

    print(www.text);
}
```

3）运行场景，发现打印出了乱码：



在 Chrome 中打开开发者工具（Windows 的快捷键为 F12），选择“Network”面板，然后输入上述网址再次发送网络请求，在请求头的响应头中可以发现该网站返回的数据是经过压缩的：



注意 我们需要对该数据进行解压缩。

3.2.2 使用 GZipStream 解压缩字符数组

在网络服务编程中，数据压缩是一个提高性能的重要手段。因为每次传输的数据越小，单次访问的处理时间就越短，在单位时间内可以完成请求数就越高，进而达到提高性能的目的。实际上就是节约了网络带宽。

作为 C 端（Client 端，也就是客户端）程序，我们需要负责解压缩这个被压缩的数据，C# 中有一个 GZipStream 类可以用于解压缩 GZip 压缩的数据。

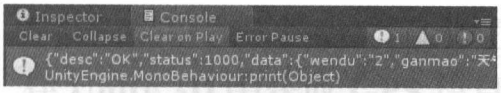
1）添加 Decompress 函数，使用 GZipStream 以解压获取的字符数据。

```
string Decompress(byte[] bytes)
{
    // Read the last 4 bytes to get the length
    // gzip appends this to the array when decompressing
    var lengthBuffer = new byte[4];
    System.Array.Copy(bytes, bytes.Length - 4, lengthBuffer, 0, 4);
    int uncompressedSize = System.BitConverter.ToInt32(lengthBuffer, 0);
    var buffer = new byte[uncompressedSize];
    using (var ms = new MemoryStream(bytes)) {
        using (var gzip = new GZipStream(ms, CompressionMode.Decompress)) {
            gzip.Read(buffer, 0, uncompressedSize);
        }
    }
    return System.Text.Encoding.UTF8.GetString(buffer);
}
```

2）在 Start 函数中调用 Decompress 函数。

```
IEnumerator Start()
{
    WWW www = new WWW( "http://wthrcdn.etouch.cn/weather_mini?city= 北京" );
    yield return www;

    string result = Decompress(www.bytes);
    print(result);
}
```

3）运行场景，可以发现结果正常。与浏览器获取的结果相同。



接收到的 JSON 字符串需要组织为特定的数据结构方便获取后在程序内使用，因此我们需要需要了解 JSON 在 Unity 中的使用方式。

3.2.3　在 Unity 中反序列化 JSON 数据

JSON 数据是一个字符串，如果仅仅通过读取字符串获取其中的数据过于烦琐又容易出错，好在 Unity 提供了一个类专门处理 JSON 数据，将它转化为一个特定数据结构的类，以便获取其中的数据。这个转化过程被称为反序列化（Deserialize）。

网络上也有在线工具可以将字符串的 JSON 数据转化为可视化的数据结构，例如 http://www.jsoneditoronline.org/ 就是一个在线 JSON 数据的编辑器。我们把刚刚获取的 JSON 数据粘贴至左侧，点击▸，在右侧可以看到这串 JSON 数据的层级关系，如下图所示：

JSON 中的数据类型有：

- float：其实就是数字，在文本中即‘:’后的数字（没有被“”包围）
- *string：字符串，在文本中即‘:’后的内容，被“”包围*
- 对象，可包含若干个字符串，数字或数组
- 数组，相同类型的连续数据，可以是上述三种类型之一

由上图可以看出，最上级的 object 是根对象（也就是所有其他数据都是它拥有的，像一个文件夹），包含 desc（字符串），status（数字），data（对象）。而 data 这个对象又包含 wendu（字符串），ganmao（字符串），forecast（数组），yesterday（对象），aqi（字符串）和 city（字符串）。

用 C# 代码表示为：

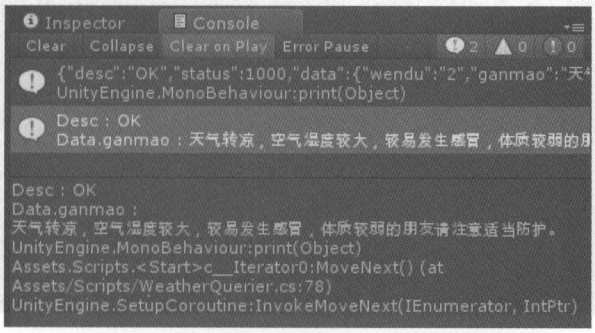
```
[System.Serializable]
public class ResponseData
{
    public string      desc;
    public int         status;
    public WeatherData data;
}

[System.Serializable]
public class WeatherData
{
    public string wendu;
    public string ganmao;
    public string aqi;
    public string city;
    public string test;

    public WeatherForecastData[] forecast;
}


[System.Serializable]
public class WeatherForecastData
{
    public string fengxiang;
    public string fengli;
    public string low;
    public string high;
    public string type;
    public string date;
}
```

在 WeatherData 中多了一个名为 test 的字符串，少了一个 yesterday。这些并不会影响序列化操作。

接下来我们在 Start 函数的最后加上反序列化处理：

```
// Deserialize JSON
ResponseData response = JsonUtility.FromJson<ResponseData>(result);
print("Desc : " + response.desc + "\n" +
      "Data.ganmao : " + response.data.ganmao);
```

运行项目，查看结果。



3.3　在 Unity 3D 中获取 GPS 信息

3.3.1　LocationService 类

在 Unity 中使用 LocationService 类可以访问设备的 GPS，该类的实例集成在 Input 类内。接口结构比较简单，成员变量包括：

- isEnabledByUser : 设备的位置服务是否已开启
- lastData : 上一次获取的地理位置信息
- status : LocationService 当前的状态，包括初始化、停止、运行、失败

成员函数包括：

- Start : 开始获取位置信息（由于 GPS 初始化比较费时，因此需要使用协程或在 Update 中等待）

Stop : 停止获取位置信息

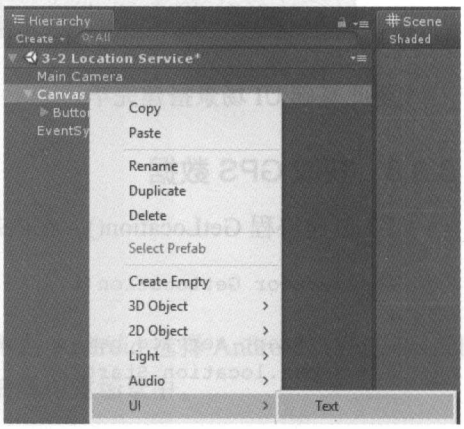
3.3.2　构建场景和 UI 处理逻辑

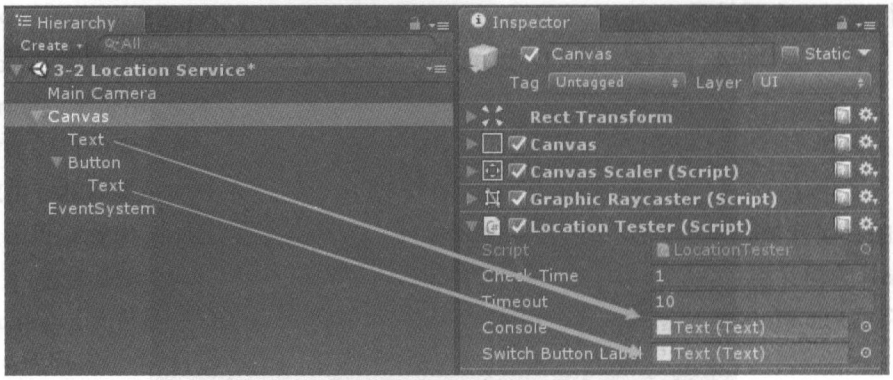
1）创建按钮，在 Hierarchy 中右键单击，依次选择 UI → Button 创建按钮，将 Button 的 Text 改为“Start”。

2）创建文本，在 Canvas 上点击右键单击，依次选择 UI → Text。

3）创建 LocationTester.cs，为其添加 Console 和 Switch Button Label 公有成员变量，类型为 Text。添加 Check Time 和 Timeout，类型为 float，用于检测 GPS 是否准备就绪。将该脚本拖至 Canvas 上。

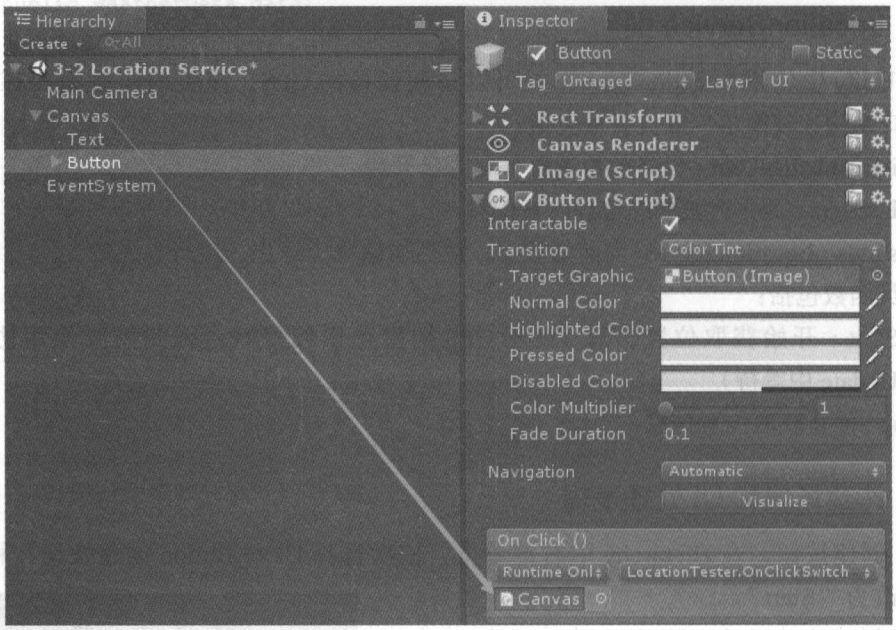
4）将创建的文本的引用赋予 Console，将按钮的 text 赋予 Switch Button Label。





5）在 LocationTester.cs 中添加按钮响应函数 OnClickSwitch()。

6）为按钮 Button 添加点击响应函数，将 Canvas 设为点击响应 GameObject，选择 LocationTester.OnClickSwitch 为回调函数。



至此基础 UI 场景搭建完毕。

3.3.3 获取 GPS 数据

1）创建协程 GetLocation()，在内部启动定位服务并等待结果。

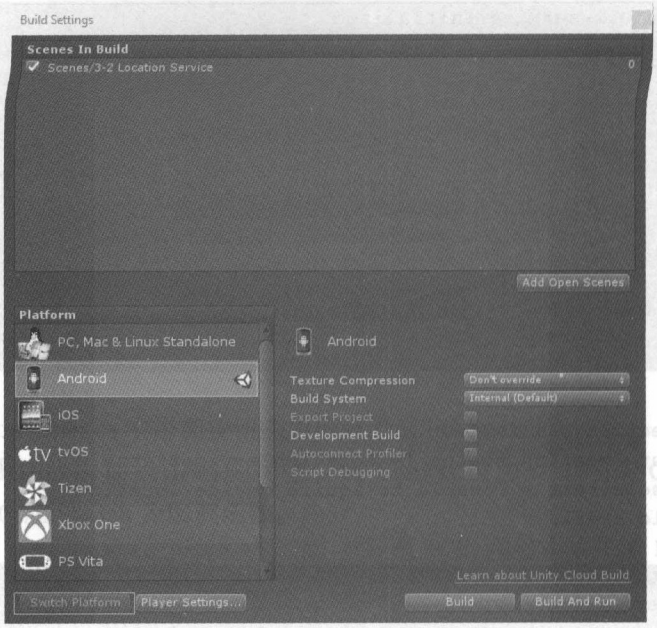
```
IEnumerator GetLocation()
{
    // Start service before querying location
    Input.location.Start();
```

```
    // Wait until service initialized
    float waitedTime = 0;
while (Input.location.status == LocationServiceStatus.Initializing
&& timeout > waitedTime) {
        yield return new WaitForSeconds(checkTime);
        waitedTime += checkTime;
    }

    // Service didn't initialize in 20 seconds
    if (timeout <= waitedTime) {
        console.text = "Location service init time out!";
        yield break;
    }

    // Connection has failed
    if (Input.location.status != LocationServiceStatus.Running) {
        console.text = "Unable to aquire device location";
        yield break;
    } else {
        // Access granted and location value could be retrieved
        console.text = "Location" + "\n" +
            "latitude  : " + Input.location.lastData.latitude  + "\n" +
"longitude : " + Input.location.lastData.longitude + "\n" +
"altitude  : " + Input.location.lastData.altitude  + "\n" +
"horizontalAccuracy : " +
Input.location.lastData.horizontalAccuracy + "\n" +
"timestamp : " + Input.location.lastData.timestamp;
    }
}
```

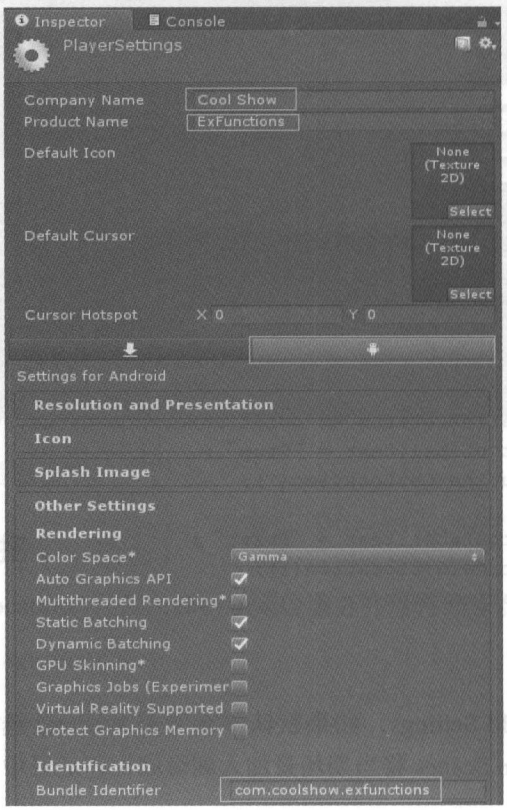
2）在按钮点击事件回调中添加 GPS 检查，并控制按钮的文本。

```
public void OnClickSwitch()
{
    // First, check if user has location service enabled
    if (!Input.location.isEnabledByUser) {
        console.text = "Location service is disabled";
        return;
    }

    if (Input.location.status == LocationServiceStatus.Stopped) {
        switchButtonLabel.text = "Stop";
        StartCoroutine(GetLocation());
    } else {
        switchButtonLabel.text = "Start";
        StopCoroutine(GetLocation());
        Input.location.Stop();
    }
}
```

3）点击 File → Build Settings，如果默认平台不是 Android 选择 Android，点击 Switch Platform。点击 Add Open Scenes 将当期场景加入到编译构建场景中。

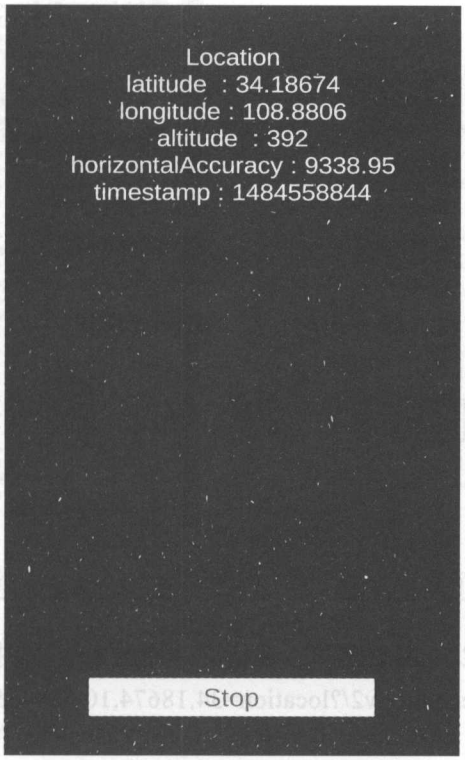


4）点击 Player Settings，填写 Company Name 和 Product Name，在安卓平台编译构建选项的 Other Settings 的 Bundle Identifier 中填入合法字符。



5）在 Build 窗口中点击 Build，填写 APK 文件名，生成 APK 文件。

6）将 APK 文件上传至 Android 手机，安装并运行，确保 GPS 运行或有 WiFi 网络连接。点击 Start，则会出现：

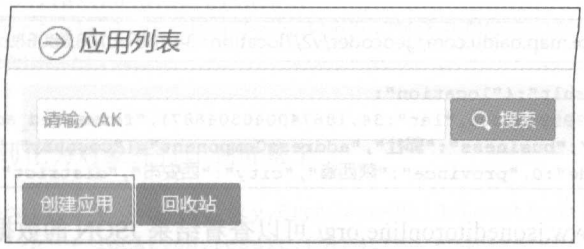


根据这些经纬度信息，我们可以获取用户所在城市。如果是网络游戏还能获取周围的玩家等。

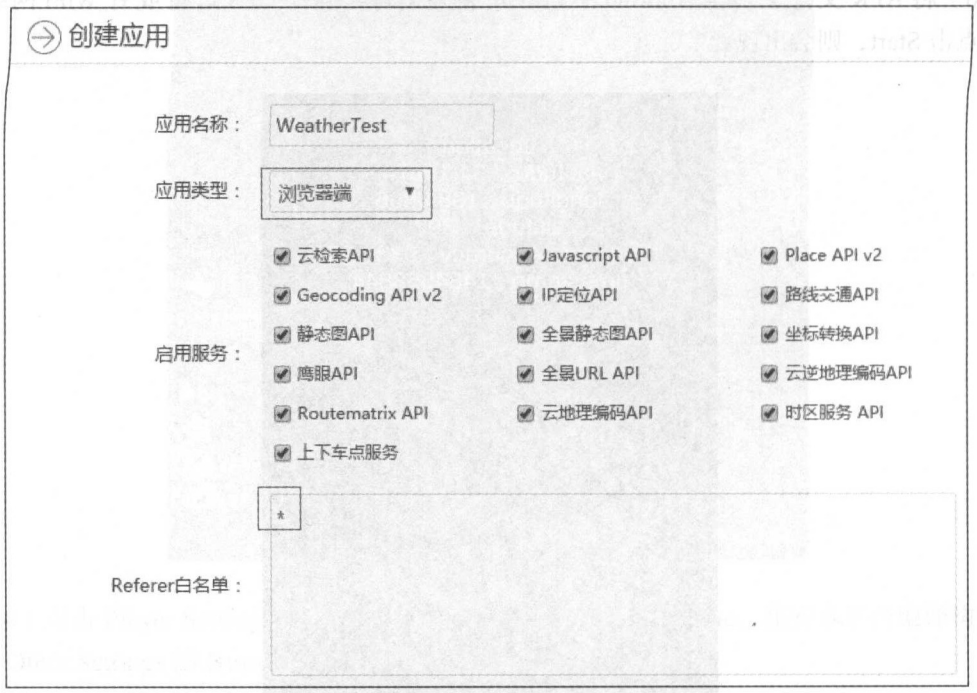
3.3.4　通过地理位置获取城市

正如上文所述，获取经纬度后可以进而得知所在地。现在各大 IT 公司提供的开发者平台都有很多有用的免费功能，位置服务自然也不例外。本书将采用百度开发者平台获取用户的所在城市。

1）登录 http://lbsyun.baidu.com/ 申请一个开发者账号，创建一个应用。

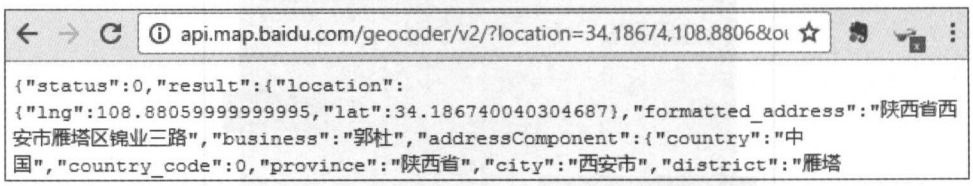


2）“应用类型”选择“浏览器端”，权限全部打开，Referer 白名单填入“*”，点击提交。



3）在浏览器中输入以下网址测试，注意将“你的 AK”替换为你刚创建应用的 AK，http://api.map.baidu.com/geocoder/v2/?location=34.18674,108.8806&output=json&pois=1&ak=你的 AK。

4）会得到一串 JSON 的结果字符串，包括了获取的相关地理信息。



5）使用 http://www.jsoneditoronline.org/ 可以查看结果 JSON 的数据结构。

6）在代码中创建和该结果数据结构相同的类（这里仅仅定义了感兴趣的数据）。

```
[System.Serializable]
public class BaiduLocation
{
    public Result result;
}

[System.Serializable]
public class Result
{
    public string formatted_address;
}
```

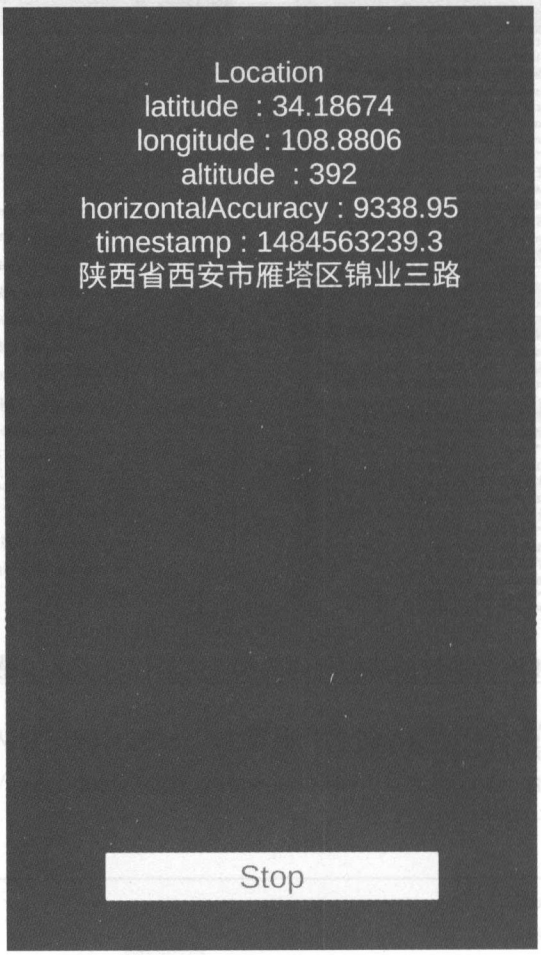
7）在 Unity 的代码中使用 WWW 获取这些数据：在 GetLocation() 成功获取经纬度后添加如下代码获取结果：

```
string url = "http://api.map.baidu.com/geocoder/v2/?location=" +
    Input.location.lastData.latitude  + "," +
    Input.location.lastData.longitude +
    "&output=json&pois=1&ak=TMaA1YxkquVMe3szPWcb4bBXyoucPBno" ;
WWW www = new WWW(url);
yield return www;
```

8）将 JSON 反序列化为对象并打印到屏幕上。

```
BaiduLocation location = JsonUtility.FromJson<BaiduLocation>(www.text);
console.text += "\n" + location.result.formatted_address;
```

9）部署到 Android 设备上，并运行，结果如下：



可以结合 3.2 节制作一个获取当地天气的小程序。

3.4 在 Unity 中实现二维码的生成与识别

3.4.1 QR CodeBarcode Scanner and Generator 简介

QR CodeBarcode Scanner and Generator 是在 Unity Store 中的一款扫描 / 生成二维码的插件，代码逻辑简易而且可以跨平台操作，如果在项目中需要实现使用扫描 / 生成二维码的功能，使用这款插件可以事倍功半。（后文中简称这款插件为 QR。）

QR 是由纯 C# 代码编写，不包含任何第三方库，简捷轻便并保证了良好的跨平台兼容性。使用中需要注意的核心类包括：

❑ QRCodeDecodeController：识别二维码的类

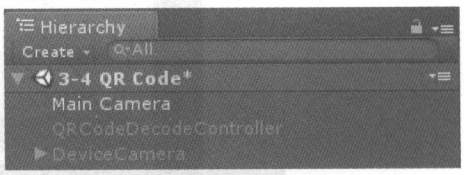
- QRCodeEncodeController：生成二维码的类
- DeviceCameraController：摄像机的控制类
- CameraPlaneController：摄像机纹理平面

插件的导入

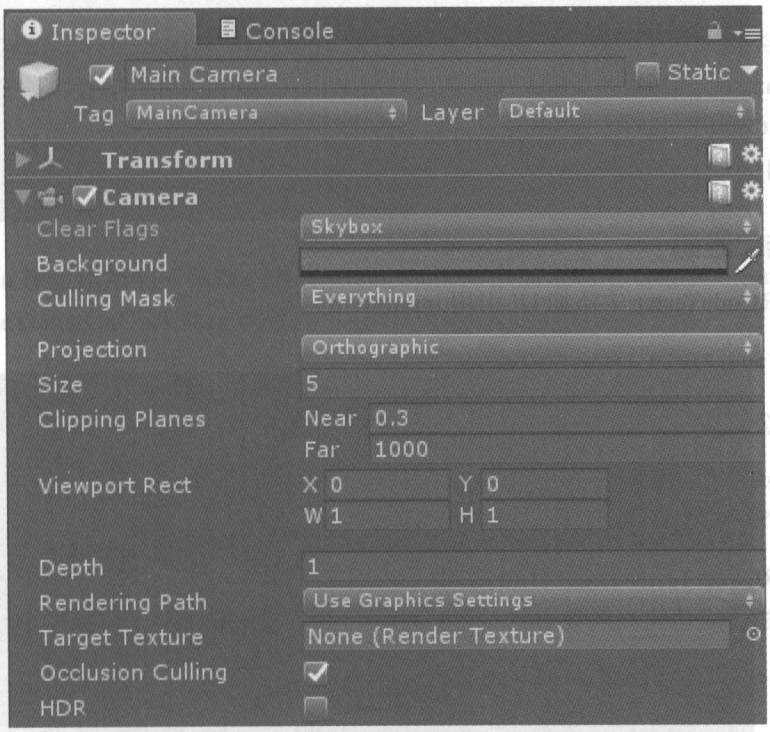
在 Unity 开启的状态下，双击“QR CodeBarcode Scanner and Generator.unitypackage”导入全部文件。注：在 Assets/QRcode 文件夹中，Prefab 文件夹名称拼错了（Perfab），将 Prefab 文件夹下的 QRController 改为 QRCodeDecodeController（这个 Prefab 是用于识别二维码的，此处名称不合适可能会给后期开发造成疑惑，但此步骤并不是必须的）。

3.4.2　搭建工程场景

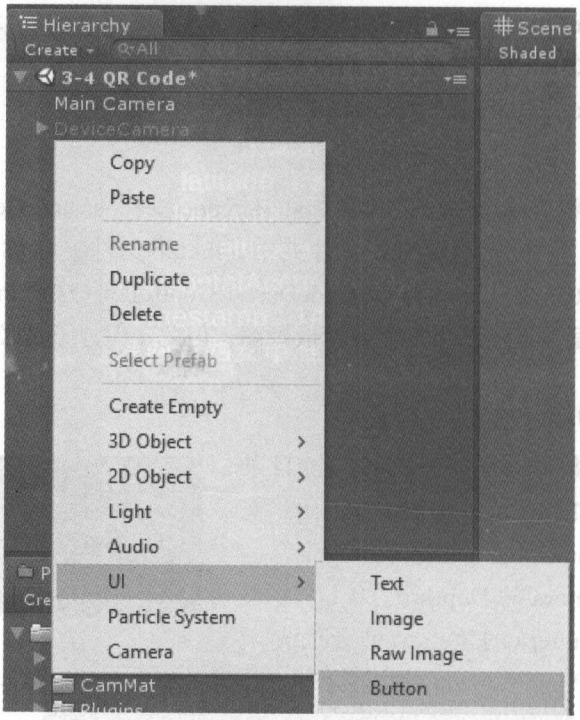
1）新建场景，将 Assets/QRcode/Prefab/ 目录下的 DeviceCamera 和 QRCodeDecodeController 拖至场景中，并将其 GameObject 设置为 Disable。



2）将 Main Camera 的 Depth 改为 1（这里是为了确定 Main Camera 的渲染结果会优先于 DeviceCamera，这样 Main Camera 的内容才能在摄像机内容之上显示）。



3）创建开始扫描按钮，在 Hierarchy 面板中点击右键，选择 UI → Button 创建按钮。调整位置和大小，并将 Text 改为 Scan（扫描二维码）。



4）使用类似上述步骤在 Canvas 下再创建一个 Generate（生成二维码）的按钮。

5）在 Canvas 下创建一个输入框，依次选择 UI → Input Field 并调整其位置和大小。

6）创建 C# 脚本 QRCodeTester.cs，将该脚本拖至 Main Camera 中。

至此项目的基本框架搭建完毕。

3.4.2 扫描二维码

1）在 QRCodeTester 中添加 UI 引用 inputField：

```
public InputField inputField;
```

2）添加公有成员 decoder 和 deviceCamera 的引用：

```
public QRCodeDecodeController decoder;
public DeviceCameraController deviceCamera;
```

3）为 QRCoderTester 添加对应的引用：

4）创建扫描成功回调函数 OnScanFinished 将结果显示到屏幕上，并停止扫描：

```
void OnScanFinished(string scanText)
{
    inputField.text = scanText;

    // Disable decoder and device camera when scan complete
    decoder.gameObject.SetActive(false);
```

```
    deviceCamera.gameObject.SetActive(false);
}
```

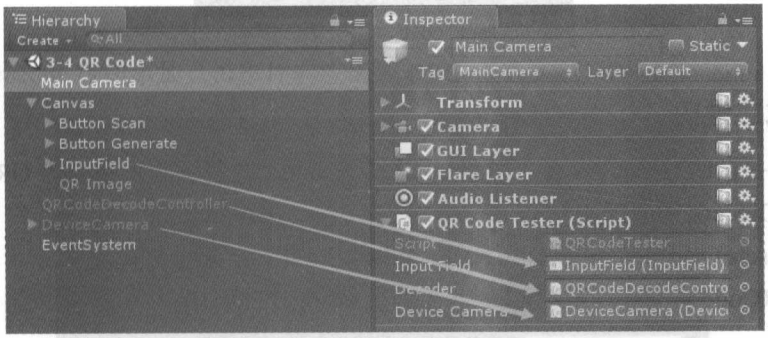
5）在 Start 函数中注册 e_QRScanFinished 事件：

```
void Start()
decoder.e_QRScanFinished += OnScanFinished;
}
```

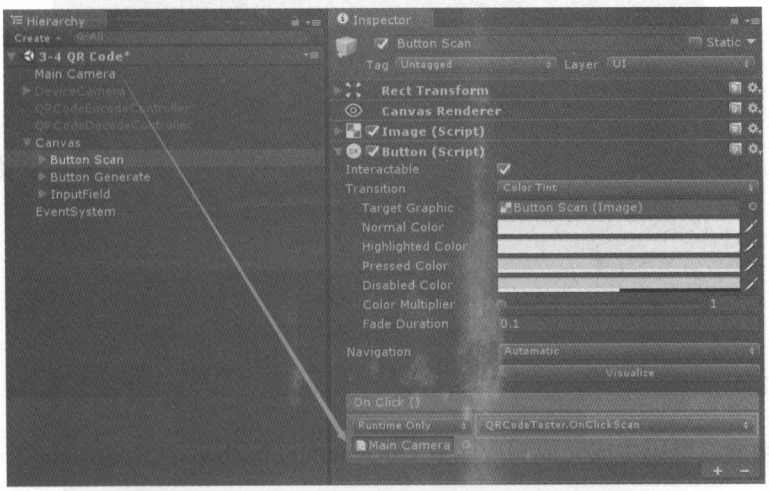
6）添加点击 Scan 按钮的响应函数 OnClickScan，在内部启动 Scan 的关联 GameObject：

```
public void OnClickScan()
{
    deviceCamera.gameObject.SetActive(true);
    decoder.gameObject.SetActive(true);
    decoder.Reset();
}
```

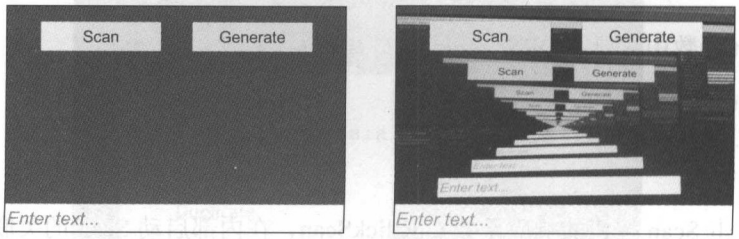
7）为 QRCodeTester 添加对应的对象引用。



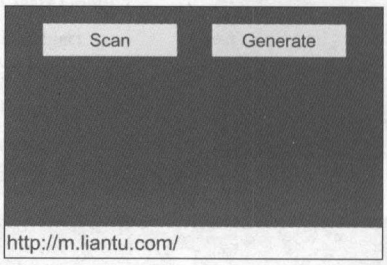
8）为 Scan 按钮添加响应函数，并将 Main Camera 添加至响应 GameObject 处，再在右侧选择 QRCodeTester.OnClickScan 函数。



9）运行场景，点击 Scan 按钮，摄像机画面出现在背景中。



10）扫描一个二维码，显示结果。



3.4.3 生成二维码

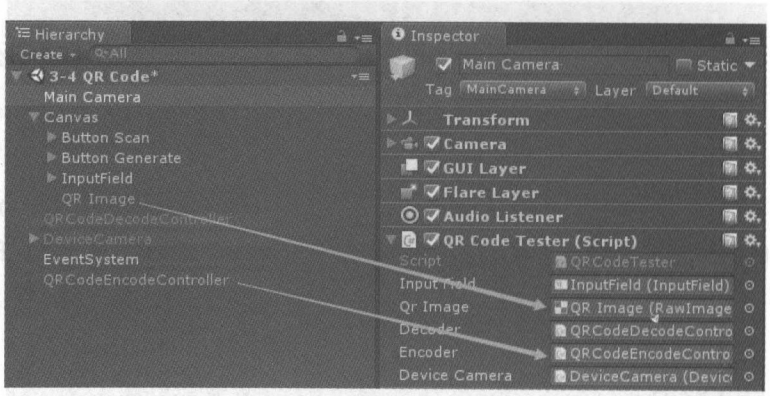
1）创建一个 UI 图片，依次选择 UI → Raw Image，并将图片重命名为 QR Image，Disable 此 GameObject。并调整其位置和大小。

2）将 Assets/QRcode/Prefab/QRCodeEncodeController 拖至场景中。

3）在 QRCodeTester.cs 添加该 QR Image 和 encoder 的引用：

```
public RawImage qrImage;
public QRCodeEncodeController encoder;
```

4）将对应 GameObject 拖至对应位置。



5）创建生成二维码生成完成响应函数 OnGenerateFinished。

```
void OnGenerateFinished(Texture2D texture)
{
    qrImage.gameObject.SetActive(true);
    qrImage.texture = texture;

    // Disable encoder and device camera when scan complete
    encoder.gameObject.SetActive(false);
    deviceCamera.gameObject.SetActive(false);
}
```

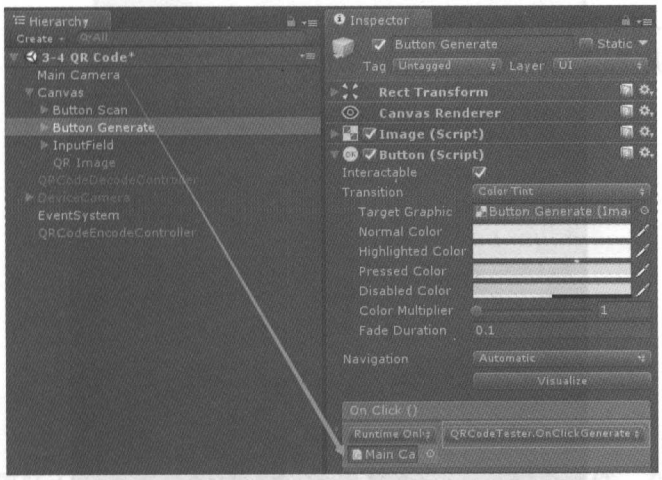
6）在 Start 函数中注册生成二维码完成事件：

```
encoder.e_QREncodeFinished += OnGenerateFinished;
```

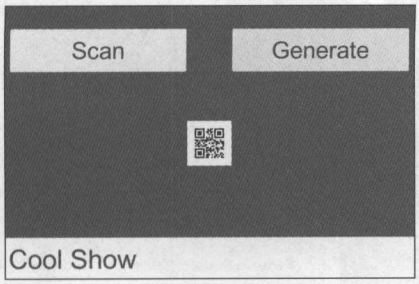
7）添加 Generate 按钮响应函数 OnClickGenerate：

```
public void OnClickGenerate()
{
    encoder.gameObject.SetActive(true);
    encoder.Encode(inputField.text);
}
```

8）为 Generate 按钮添加点击事件响应函数 QRCodeTester.OnClickGenerate。



9）运行项目，在输入框内输入任意内容后，点击 Generate，会生成对应二维码。



3.5 在 Unity 3D 中实现动态资源加载

3.5.1 AssetBundle 简介

AssetBundle 是 Unity 提供的一种开发者可以动态加载的资源类型，大多用于规模较大，或是需要进行网络更新的游戏。开发者可以根据项目需求构建一套十分灵活的资源管理机制。

Resources 文件夹虽然也可以提供动态加载的功能，但是由于该文件夹下的内容是随着项目的构建一并打包到工程中的。而且，官方的建议是不要使用。原因如下：

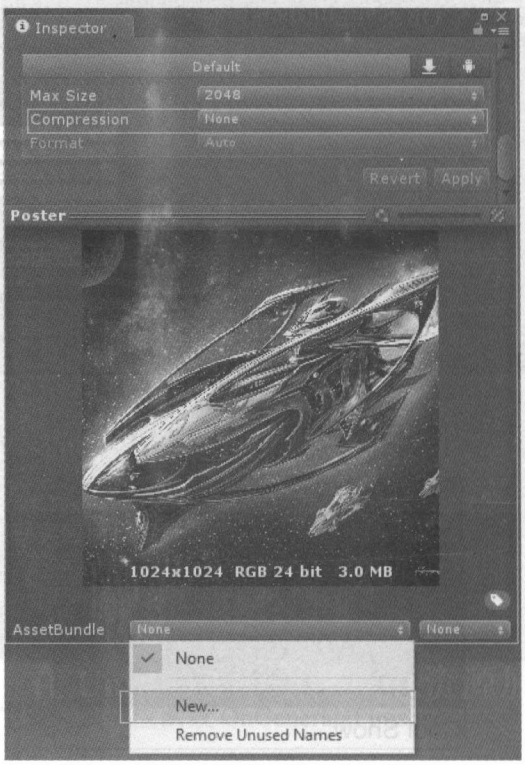
- ❑ 滥用 Resources 文件夹会导致启动时间变长，因为 Recourses 文件夹下的内容是随时加载的。
- ❑ 会导致 Unity 内部的小粒度内存管理变得十分困难。
- ❑ 随着 Resources 文件夹数量的增加，在这些文件夹中管理资源会变得十分困难。
- ❑ 无法更新内容。

因此熟练地使用 AssetBundle 对于大型项目中的资源管理来说变得尤为重要。本节重点介绍如何从网络服务器上获取资源并加载至 Unity 中。

3.5.2 如何创建 AssetBundle

本书是基于 Unity 5.x 的 AssetBundle 制作的项目，Unity 4.x 的版本可能会略有不同。

1）将测试图片 Poster.jpg 放入 Assets/Textures 文件夹下。



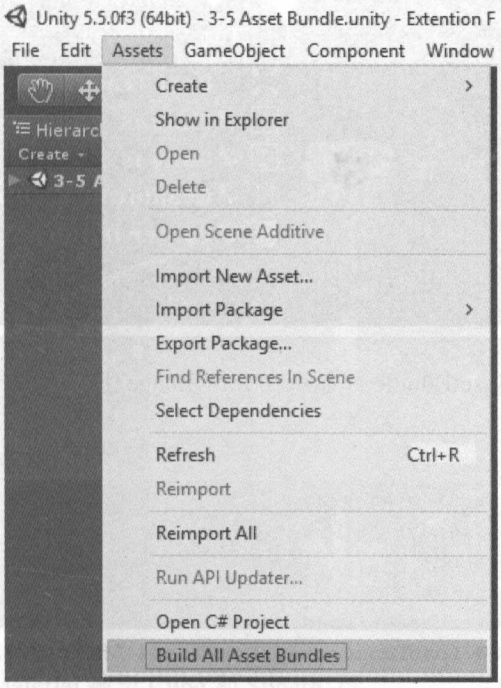
2）将 AssetBundle 展开，选择 New，输入“ui/poster”，点击回车确认。

3）在 Assets 目录下创建文件夹 Editor，在该文件夹下创建编辑器类 AssetBuilder.cs：

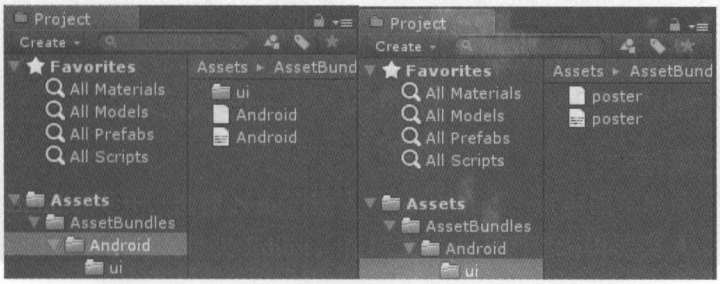
```
[MenuItem("Assets/Build All Asset Bundles")]
static void BuildAll()
{
    BuildPipeline.BuildAssetBundles("Assets/AssetBundles/Android",
BuildAssetBundleOptions.None, BuildTarget.Android);
}
```

4）在项目的 Assets 目录下创建 AssetBundles/Android 文件夹。

5）回到 Unity，会发现菜单栏，Assets 最下面多了一个 Build All Asset Bundles 的选项，点击该选项。



可以发现创建好了 poster 的 AssetBundle 和 manifest（清单）文件，在 Android 目录下建立了 Android 和 manifest 文件，用于记录该文件夹内的依赖关系。



6）启动 Nginx 服务器（参考 3.1.2 节），将 AssetBundles 文件夹移至 html 文件夹下。

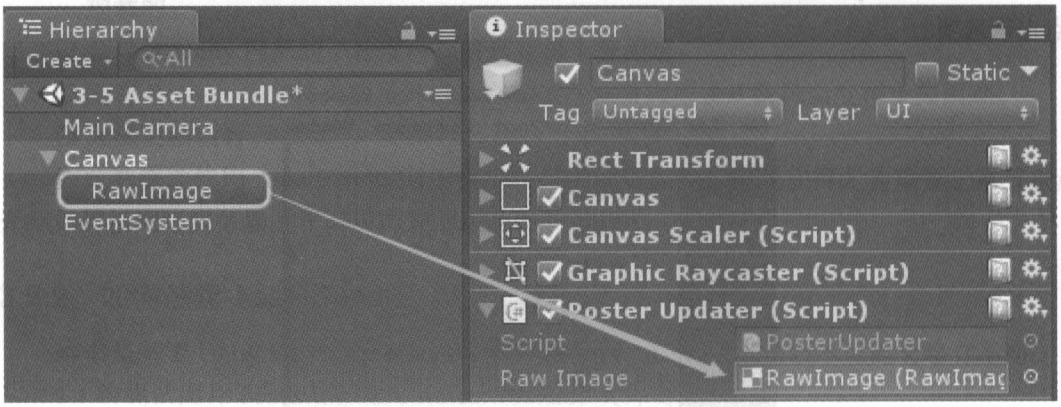
3.5.3 如何加载 AssetBundle

1）创建一张 UI 图片：在 Hierarchy 面板中点击右键，依次选择 UI → Raw Image，调整图片的位置和大小。

2）创建海报更新脚本 PosterUpdater.cs，添加 Raw Image 的引用：

```
public RawImage rawImage;
```

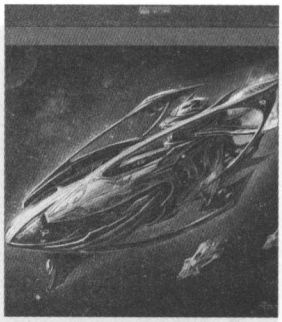
3）将脚本拖至 Canvas，并在 Inspector 中添加引用。



4）在 Start 中获取 AssetBundle 并显示在 Raw Image 上。

```
IEnumerator Start()
{
    // Wait until caching is ready
    while (!Caching.ready)
        yield return null;

    // Download poster asset bundle
     var www = WWW.LoadFromCacheOrDownload("http://127.0.0.1/AssetBundles/
Android/ui/poster", 0);
    yield return www;
    if (!string.IsNullOrEmpty(www.error)) {
        Debug.Log(www.error);
        yield break;
    }

    // Get texture from asset bundle
    Texture2D texture = www.assetBundle.LoadAsset<Texture2D>("Poster");
    if (texture != null) {
        rawImage.texture = texture;
    }
}
```

5）运行项目查看结果。



上述处理中，WWW.LoadFromCacheOrDownload 函数会优先在缓存中寻找该文件是否已经存在，若存在就会从缓存中直接读取，并不会访问网络。而且在第一次获取该资源后会自动写入缓存，这部分的缓存大小最大为 4GB，可以通过 Caching 类查询或设置（Caching.maximumAvailableDiskSpace），这也是 Unity 官方推荐的加载方式。

使用 WWW 对象虽然也可以获取网络对象，但没有缓存机制，而且伴有内存的额外使用。

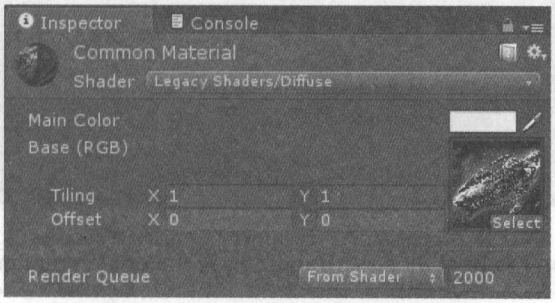
3.5.4　AssetBundle 之间的依赖关系

在实际使用中，几个 Prefab 常常会使用到一些相同的资源，当这些资源没有在同一个 AssetBundle 中时，就需要考虑载入的顺序。例如，一个 Cube 和一个 Sprite 使用同一张 Texture 时，就会出现这种问题。

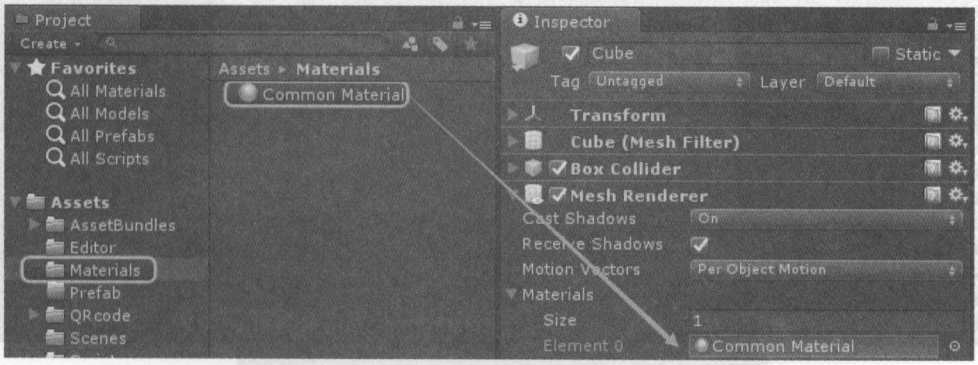
我们接着上面的项目来模拟这个使用场景。

1）在场景中各创建一个 Cube、Sphere 和 Directional Light。

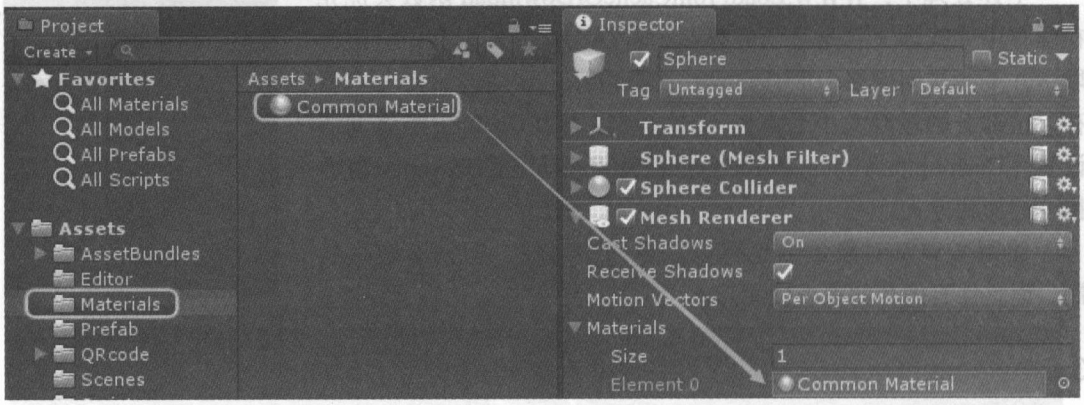
2）在 Assets 文件夹下创建 Materials 文件夹，创建一个 Common Material 的材质，并将材质选为 Legacy Shaders/Diffuse，将 Textures/Poster 拖至 Texture 处。



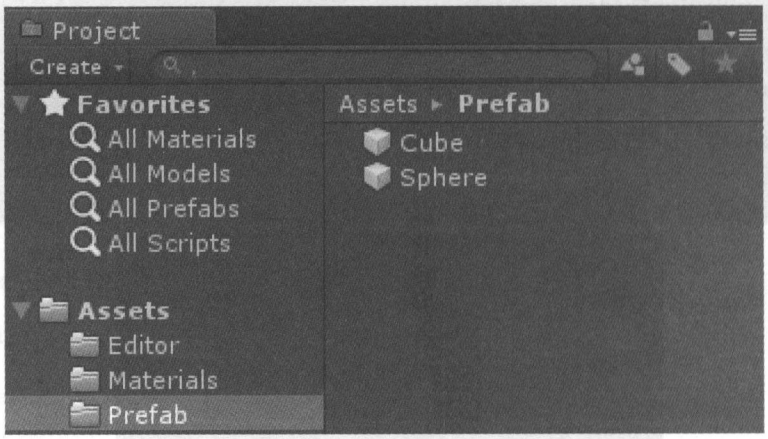
3）将材质 Common Material 拖至 Cube 和 Sphere 上。



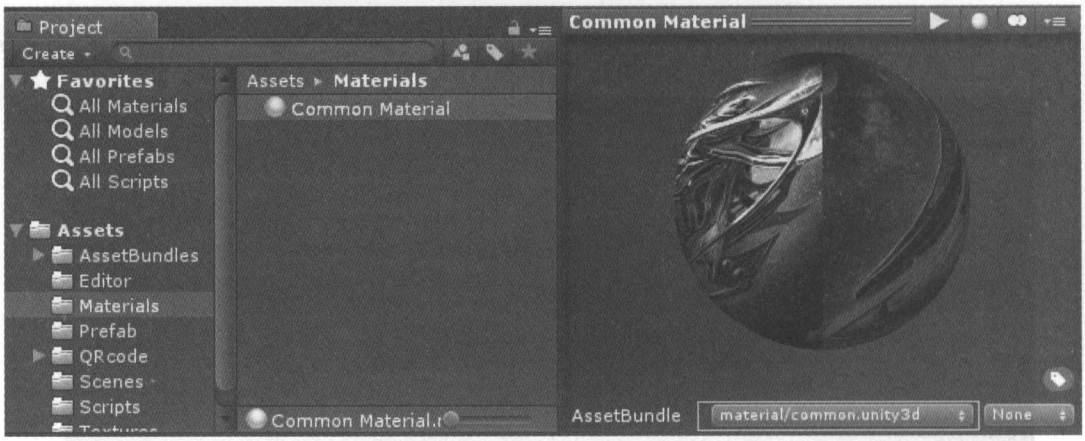
4）将材质 Common Material 拖至 Sphere 上。



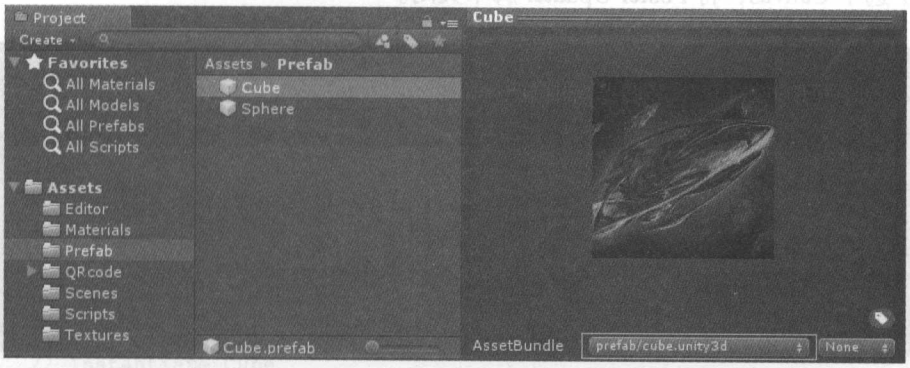
5）在 Assets 文件夹下创建 Prefab 文件夹，将 Cube 和 Sphere 拖至该文件夹创建 Prefab。



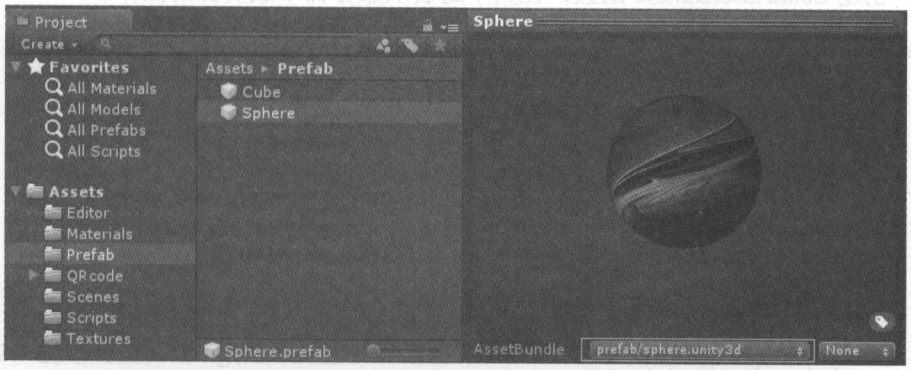
6）将材质 Common Material 的 AssetBundle 设为 material/common.unity3d。



7）将 Cube 的 Prefab 选中，将 AssetBundle 命名为 prefab/cube.unity3d。



8）将 Sphere 的 Prefab 选中，将 AssetBundle 命名为 prefab/sphere.unity3d。



9）删除场景中的 Sphere 和 Cube。

10）在 Assets 下创建 AssetBundles/Android。

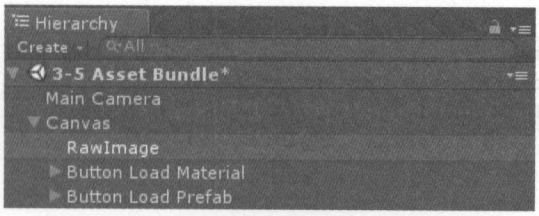
11）在文件夹在菜单栏中选择 Assets/Build All Asset Bundles。

12）进入 Assets/AssetBundles/Android 目录，可以通过文件大小看出 Texture 文件被打包到了 Material 中，BuildPipeline.BuildAssetBundles 创建 AssetBundles 时，会自动解析资源之间的依赖关系并将它们打包（Unity 5.0 版本后的新特性，4.x 并不支持这个功能）。

13）将 Assets/AssetBundles 文件夹移动至 Nginx 的 html 目录。

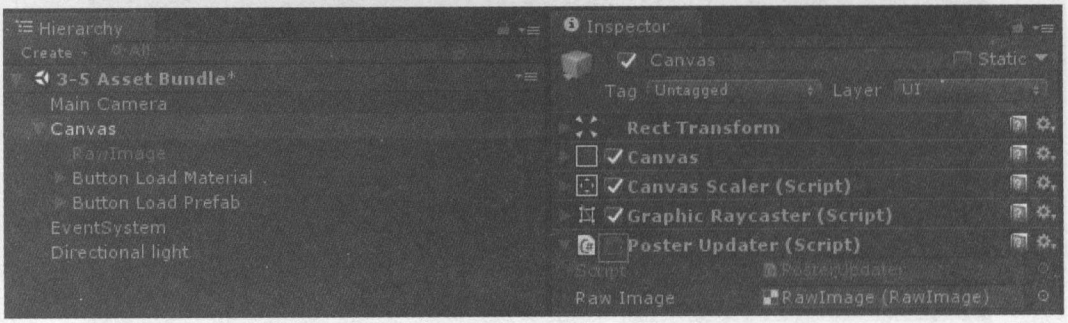
14）确定 Nginx 是启动状态。

15）在 Canvas 下创建 UI 按钮 Load Material（载入材质）和 Load Prefab（载入预制件）。在 Canvas 上右键单击，依次选择 UI → Button，调整其位置和大小。



16）关闭 Canvas 中的 Raw image。

17）选中 Canvas，将 Poster Updater 脚本关闭。



18）创建脚本 DependentAssetBundleLoader.cs，并将其拖至 Canvas 面板。

19）创建 LoadAssetBundle 函数，在其中封装从服务器下载资源的操作。

```
IEnumerator LoadAssetBundle(string url, string assetName, int version = 0)
{
    // Wait until caching is ready
    while (!Caching.ready)
        yield return null;

    // Download poster asset bundle
    var www = WWW.LoadFromCacheOrDownload(url, version);
    yield return www;
    loading = false;
    if (!string.IsNullOrEmpty(www.error)) {
        Debug.Log(www.error);
        yield break;
    }

    yield return www.assetBundle.LoadAsset(assetName);
}
```

20）创建 LoadMaterial 函数，执行载入材质操作。

```
IEnumerator LoadMaterial()
{
    // Load material from server/cache
     IEnumerator ie = LoadAssetBundle("http://127.0.0.1/AssetBundles/Android/
material/common.unity3d", "Common Material");
    while (ie.MoveNext()) {
        yield return null;
    }

    // Check asset
    Material material = ie.Current as Material;
    if (material != null) {
        print("Load material complete!");
    } else {
```

```
            print("Load material failed!");
        }
    }
```

21）创建 LoadPrefab 函数，执行载入 Prefab 操作。

```
    IEnumerator LoadPrefab()
    {
        // Load cube from server/cache
         IEnumerator ie = LoadAssetBundle("http://127.0.0.1/AssetBundles/Android/
prefab/cube.unity3d", "Cube");
        while (ie.MoveNext()) {
            yield return null;
        }
        // Instantiate Cube
         GameObject cube = Instantiate(ie.Current as GameObject, new Vector3(0, 1,
0), Quaternion.identity);

        // Load sphere from server/cache
        ie = LoadAssetBundle("http://127.0.0.1/AssetBundles/Android/prefab/sphere.
unity3d", "Sphere");
        while (ie.MoveNext()) {
            yield return null;
        }
        // Instantiate Sphere
         GameObject sphere = Instantiate(ie.Current as GameObject, new Vector3(0,
-1, 0), Quaternion.identity);
    }
```

22）创建变量 loading 成员变量表明当前是否在进行载入操作。

```
    private bool loading;
```

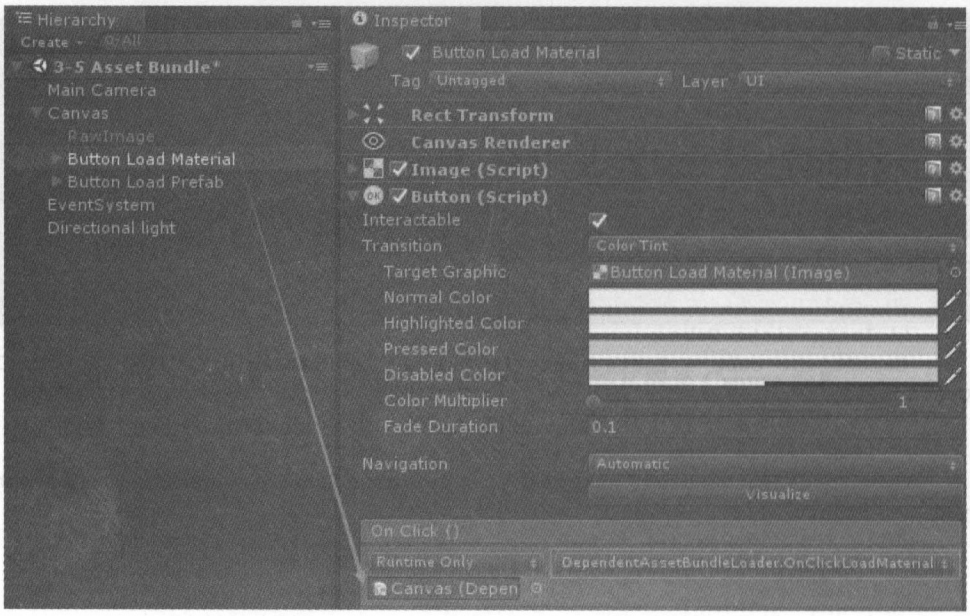
23）创建点击事件响应函数 OnClickLoadMaterial 执行载入材质操作。

```
    public void OnClickLoadMaterial()
    {
        if (!loading) {
            loading = true;
            StartCoroutine(LoadMaterial());
        }
    }
```

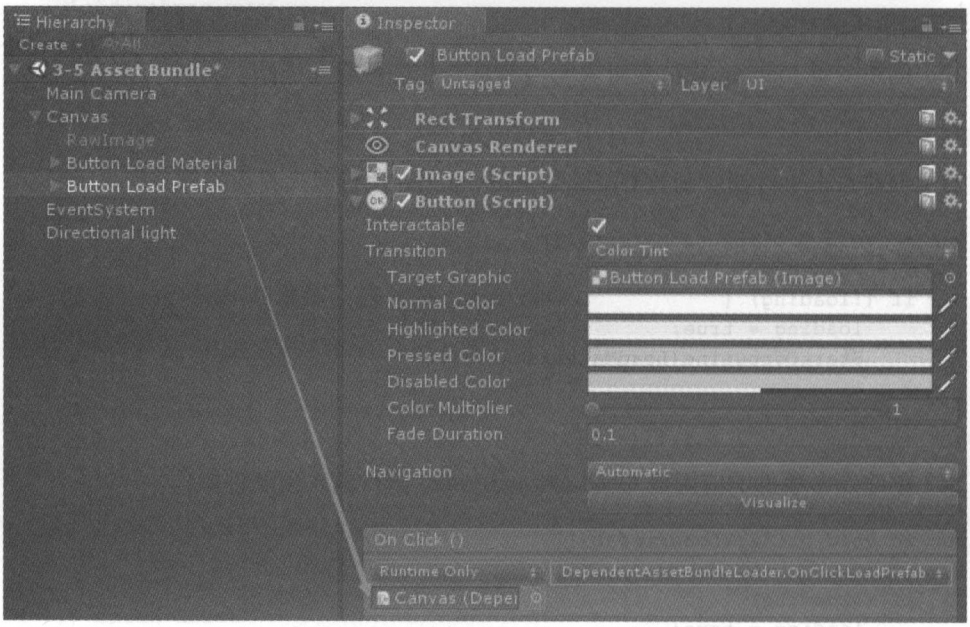
24）创建点击事件响应函数 OnClickLoadPrefab 执行载入预制件操作。

```
    public void OnClickLoadPrefab()
    {
        if (!loading) {
            loading = true;
            StartCoroutine(LoadPrefab());
        }
    }
```

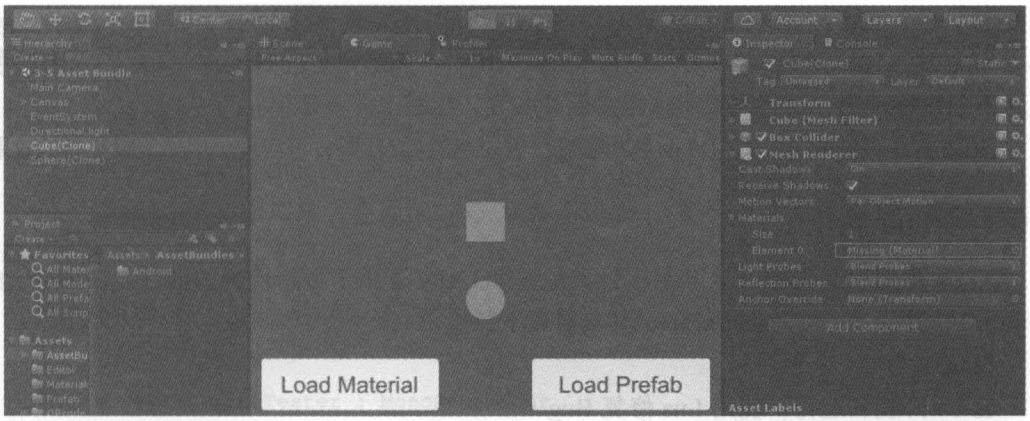
25）为按钮 Button Load Material 添加响应函数 DependentAssetBundleLoader. OnClickLoadMaterial。



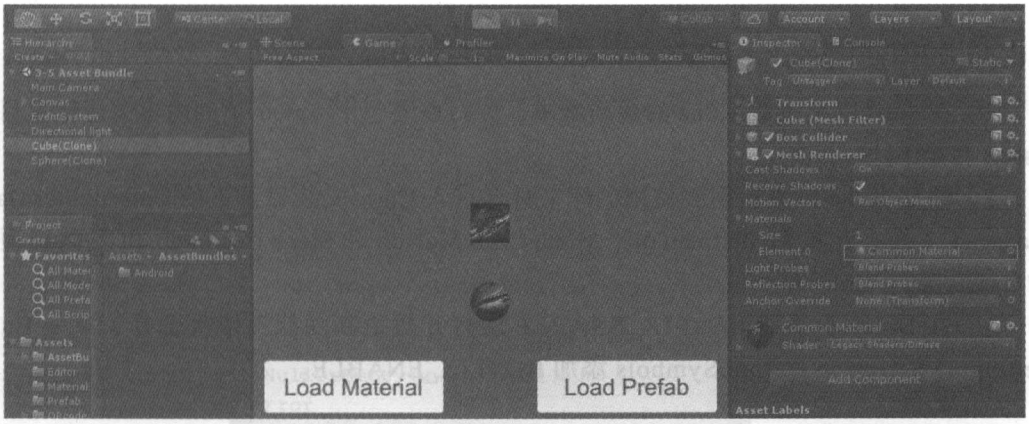
26）为按钮 Button Load Prefab 添加响应函数 DependentAssetBundleLoader. OnClickLoadPrefab。



27）运行项目，直接点击 Load Prefab 按钮，可以发现生成的两个物体没有材质，选中观察会发现材质丢失。



28）重新运行项目，先点击 Load Material 按钮，再点击 Load Prefab 按钮，可以发现显示正常。



从上面的实验可知，有时我们出于节约空间的考虑会将共用的资源单独打包，当载入时需要按照依赖的顺序依次加载，否则就会出现引用丢失的情况。

在实际项目中，当一个资源的生命周期结束后，需要释放其所占用的内存，可以通过 AssetBundle.Unload 实现。在开发作业中如果有需要研究具体 AssetBundle 的内存模型请参考 Unity 官方的解释：https://unity3d.com/cn/learn/tutorials/topics/best-practices/guide-assetbundles-and-resources?playlist=30089。

3.6　在 Unity 3D 中实现热更新

3.6.1　热更新方案比较

在应用开发中，有时我们需要更新一些逻辑代码，例如功能追加、BUG 修正等。但是 Unity 的 AssetBundle 中是不存在代码数据的，仅仅是序列化的资源。所以如何更新逻辑代

码变成了一个难题。

官方推荐过使用 C# 的反射机制将 MonoBehavior 绑定到对应 GameObject 上，但是使用反射不但步骤烦琐，而且需要引入很多额外的库，性能也并不理想。

因此 XLua 应运而生，使用 Lua 脚本更新已经现存的 C# 代码，可实现小规模逻辑更新的效果。

3.6.2 XLua 简介

XLua 是腾讯公司的一个开源项目，是一个 Unity 下用于更新代码的插件。GitHub 地址为 https://github.com/Tencent/xLua，其具有以下特点：

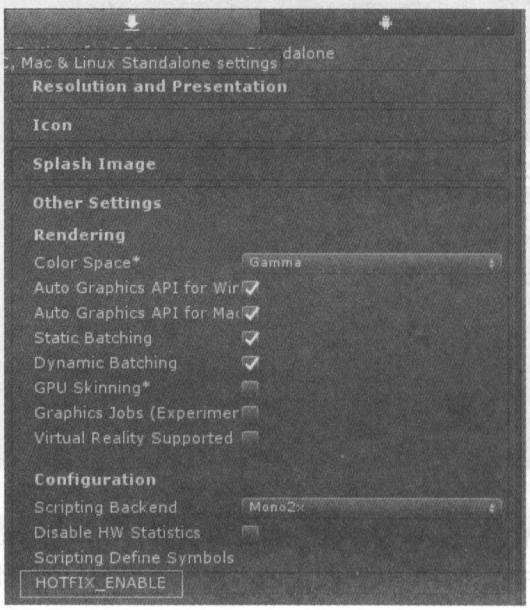
- ❑ 支持 Unity 全平台
- ❑ C# 开发和运行，使用 Lua 修复 Bug
- ❑ 功能强大，可以在运行时将 C# 中的方法、操作符、属性、事件、构造函数、析构函数和泛型替换为 Lua
- ❑ 性能不及 C#，但是在大部分场景下不会成为瓶颈

3.6.3 如何使用 XLua 更新

1）在 GitHub 上下载 XLua 的最新版本，并将压缩包中 Assets 下的内容解压至项目的 Assets 目录下。

2）添加 XLua 热更新需要使用的库文件。进入 Unity 安装目录下的 Unity\Editor\Data\Managed 文件夹，将 Mono.Cecil.dll、Mono.Cecil.Mdb.dll 和 Mono.Cecil.Pdb.dll 复制到项目的 Assets\XLua\Src\Editor 目录下。

3）在 Unity 中添加 HOTFIX_ENABLE 宏打开热更新功能：依次选择 File → Build Setting → Scripting Define Symbols 添加 HOTFIX_ENABLE。



注意　添加完成后按回车确定，确定每个需要的平台都添加了该宏。

4）新建场景，确认 Main Camera 是开启状态。

5）创建脚本 PatchTester.cs，将其拖至 Main Camera 上。

6）为 PatchTester 添加命名空间 XLua，属性 [Hotfix]（表明该类是可以被 XLua 动态更新的，支持 XLua 的函数覆盖功能）。

```
using XLua;

[Hotfix]
public class PatchTester : MonoBehaviour
```

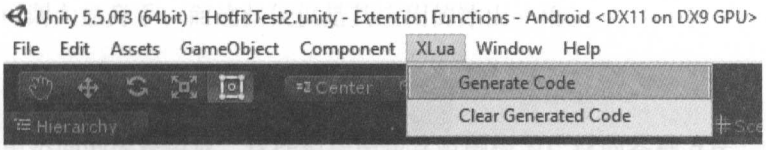
7）在 PatchTester 中添加运算处理函数 Add，这里故意写成减法制造了一个 Bug。

```
int Add(int a, int b)
{
    return a - b;
}
```

8）在 Start 函数中检查是否有 Bug 修正的 Lua 代码，如果没有，打印下载结果，如果有，执行 Bug 修正代码。

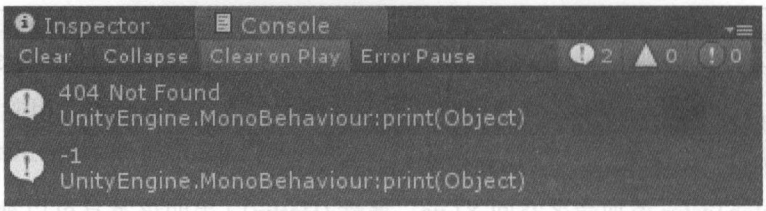
```
IEnumerator Start()
{
    // Try to download patch code
    WWW www = new WWW("http://127.0.0.1/PatchTester.lua");
    yield return www;

    if (!string.IsNullOrEmpty(www.error)) {
        // print error
        print(www.error);
    } else {
        // Execute patch when there is something to update
        luaEnv.DoString(www.text);
    }

    print(Add(1, 2));
}
```

9）在菜单栏中点击 XLua → Generate Code 生成代码。



10）启动 Nginx 服务器，确定 html 文件夹中没有 PatchTester.lua 文件。

11）运行项目，打印出下载错误，并显示 1+2 的结果为 -1，程序出现 Bug。



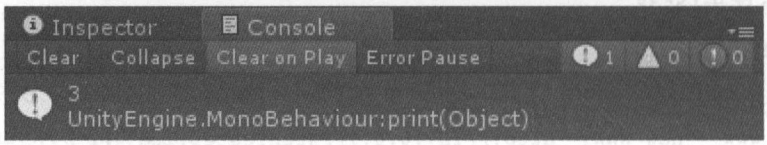
12）编辑 PatchTester.lua 的 Lua 代码。

```
xlua.hotfix(CS.Assets.Scripts.PatchTester, 'Add', function(self, a, b)
    return a + b
end)
```

这是一段简单的 Lua 代码，调用 xlua 的 hotfix 函数，第一个参数为需要修正的类名（包含其命名空间），在这个例子中是本类，但是其实可以是任何有 [Hotfix] 属性的类。第二个参数为需要修复的函数。第三个函数为需要修复的函数体。其中的 self 相当于 C# 中的 this（如果修复的是成员函数需要这个变量调用其他成员函数），后面的 a 和 b 对应函数的输入参数。

13）将 PatchTester.fix 放入 Nginx 的 html 文件夹中。

14）运行项目，可以发现运行结果正确，Lua 的补丁被打入的 C# 代码中。



使用 XLua 可以通过热更新非常便利地修复一些简单的 Bug，而不用更新 Unity 应用本身，是一种轻量级的更新方式。建议使用 XLua 修复一些性能不敏感的 Bug，在应用更新时再替换回 C# 代码。

具体的 Lua 语法类似 JavaScript，十分容易上手，语法细节可以参考 XLua 提供的例子程序，位于 Assets/XLua/Examples。

第 4 章 Chapter 4

使用 OpenCV 开发图像识别应用

4.1 OpenCV 图像识别简介

4.1.1 OpenCV 图像识别技术应用领域

OpenCV 是一个用于图像处理、分析、机器视觉学习方面的开源函数库。无论你是做科学研究，还是商业应用，OpenCV 都可以作为你理想的选择，因为它具有良好的可移植性，而且对于以上两者，它完全是免费的。

该库采用 C 及 C++ 语言编写，可以在 Windows、Linux、Mac OSX 上运行。该库的所有代码都经过官方优化，计算效率很高，因为，它更专注于设计成为一种用于实时系统的开源库。OpenCV 采用 C 语言进行优化，而且，在多核机器上，其运行速度会更快。它的一个目标是提供友好的机器视觉接口函数，从而降低复杂的机器视觉产品开发成本，加速产品面世。该库包含了横跨工业产品检测、医学图像处理、安防、用户界面、摄像头标定、三维成像、机器视觉等领域的超过 500 个接口函数。

同时，由于计算机视觉与机器学习密不可分，OpenCV 也包含了比较常用的机器学习算法。或许，很多人知道，图像识别、机器视觉在安防领域有所应用。但很少有人知道，航拍图片和街道图片更严重依赖于机器视觉的摄像头标定、图像融合等技术。

近年来，在入侵检测、特定目标跟踪、目标检测、人脸检测、人脸识别、人脸跟踪等领域，OpenCV 可谓大显身手，而这些仅仅是其应用的冰山一角。

4.1.2 OpenCV 技术模块简介

OpenCV 由不同的模块组成，这些模块中包含范围极为广泛的各种方法，从底层的图像颜色空间转换到高层的机器学习工具。

下面是在官方文档中列出的最重要的模块，这里只做简单介绍，后续详解。

- core：简洁的核心模块，定义了基本的数据结构，包括稠密多维数组 Mat 和其他模块需要的基本函数。
- imgproc：图像处理模块，包括线性和非线性图像滤波、几何图像转换（缩放、仿射与透视变换、一般性基于表的重映射）、颜色空间转换、直方图等。
- video：视频分析模块，包括运动估计、背景消除、物体跟踪算法。
- calib3d：包括基本的多视角几何算法、单体和立体相机的标定、对象姿态估计、双目立体匹配算法和元素的三维重建。
- features2d：包含显著特征检测算法、描述算子和算子匹配算法。
- objdetect：物体检测和一些预定义的物体的检测（如人脸、眼睛、杯子、人、汽车等）。
- ml：多种机器学习算法，如 K 均值、支持向量机和神经网络。
- highgui：一个简单易用的接口，提供视频捕捉、图像和视频编码等功能，还有简单的 UI 接口（iOS 上可用的仅是其一个子集）。
- gpu：OpenCV 中不同模块的 GPU 加速算法（iOS 上不可用）。
- ocl：使用 OpenCL 实现的通用算法（iOS 上不可用）。

除上述模块外，还有一些其他辅助模块，如 Python 绑定和用户贡献的算法。

基础类和操作

OpenCV 包含几百个类。为简便起见，我们只看几个基础的类和操作，了解这几个核心类应该足以对这个库的机理产生一些感性认识。

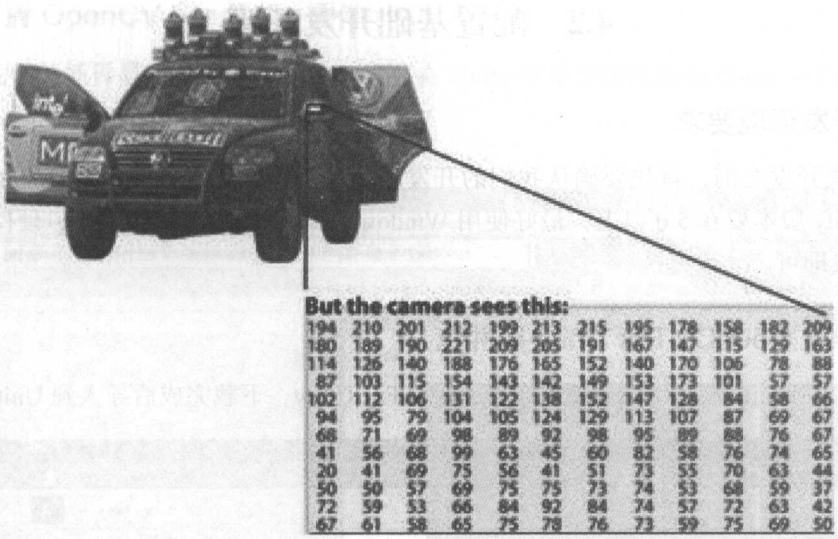
Mat 类是 OpenCV 的核心数据结构，用来表示任意 *n* 维矩阵。因为图像只是二维矩阵的一个特殊场景，所以也使用 Mat 来表示。也就是说，Mat 将是你在 OpenCV 中最常使用的类。

一个 Mat 类中包含着描述图像格式的信息。图像数据只是被引用，并能为多个 Mat 实例共享。OpenCV 使用类似于 ARC 的引用计数方法，以保证当最后一个来自 Mat 的引用也消失的时候，图像数据会被释放。图像数据本身是图像连续的行的数组（对 *n* 维矩阵来说，这个数据是由连续的 *n*–1 维数据组成的数组）。使用 step[] 数组中包含的值，图像的任一像素地址都可通过下面的指针运算得到：

```
uchar *pixelPtr = cvMat.data + rowIndex * cvMat.step[0] + colIndex * cvMat.
step[1]
```

每个像素的数据格式可以通过 type() 方法获得。除了常用的每通道 8 位无符号整数的灰度图（1 通道，CV_8UC1）和彩色图（3 通道，CV_8UC3），OpenCV 还支持很多不常用的格式，例如 CV_16SC3（每像素 3 通道，每通道使用 16 位有符号整数），甚至 CV_64FC4（每像素 4 通道，每通道使用 64 位浮点数）。

如上图所示，在标出的镜子区域中你见到的只是一个矩阵，该矩阵包含了所有像素点的强度值。如何获取并存储这些像素值由我们的需求而定，最终在计算机世界里所有图像都可以简化为数值矩阵以及矩阵信息。作为一个计算机视觉库，OpenCV 的主要目的就是通过处理和操作这些信息，来获取更高级的信息。因此，OpenCV 如何存储并操作图像是你首先要学习的。



FaceTracker 类是 OpenCV 中实现人脸识别的关键类，用于存储矩阵中跟踪面部信息的存储与读取。

cvPoint 类可以对 Mat 矩阵点位置信息进行存储与读取操作，在矩阵点与世界坐标的转换中起着关键的作用。

4.1.3　OpenCV For Unity 插件介绍

OpenCV For Unity 这款插件主要是将 OpenCV 经过转换打包成 Unity 插件包，在插件中添加了一些类库与人类识别校验文件库，我们可以很方便地调用。完成一些我们想要的功能，但是这款插件是收费的，需要花 95 美元购买才可以使用。

在 Unity 中可以使用 C# 编程语言调用 OpenCV 类库中的任意函数与算法，而实现在 Unity 中开发图像识别程序，这款插件还提供了一些实例工程，我们可以了解一些功能在 Unity 中的实现方法。Unity 有很强大的跨平台性，在开发完应用后我们可以将应用发布在很多主流平台。

这款插件在 Unity Asset Store 中可以下载到，OpenCV For Unity 是一个基本的类库，在 Unity Asset Store 也可以找到基于这款插件衍生出来的应用实例供我们学习。

支持平台包括：

- ❑ EditorPreview
- ❑ Windows
- ❑ Mac
- ❑ Linux
- ❑ iOS
- ❑ Windows StoreApp 8.1
- ❑ WindowsPhone 8.1
- ❑ Windows 10 UWP（测试版）
- ❑ WebGL（测试版）

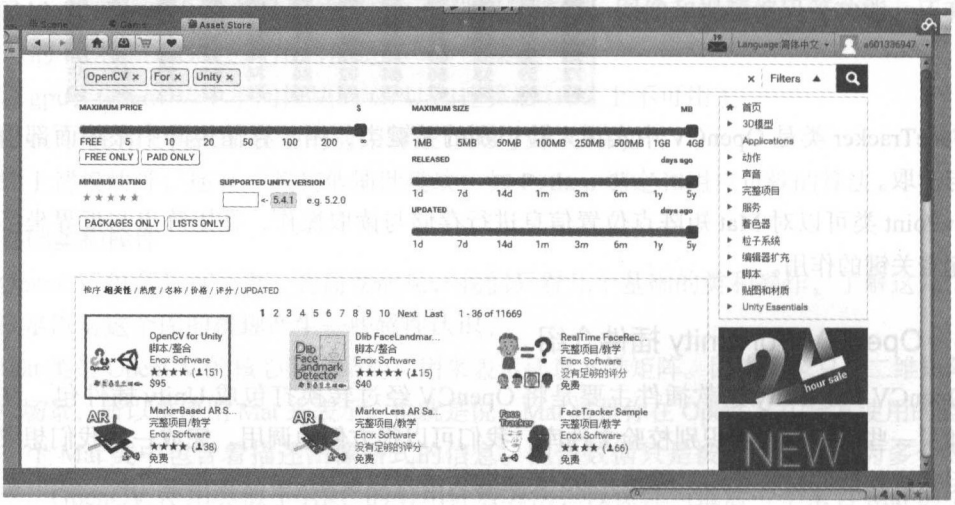
4.2 配置基础开发环境

4.2.1 开发环境要求

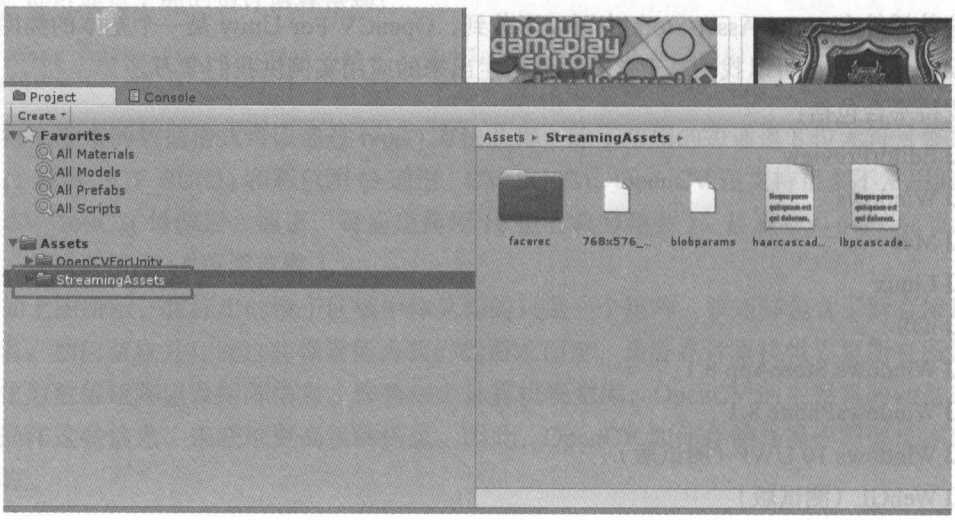
在正式开发之前，首先要确认我们的开发环境是否符合要求，在开发 OpenCV 应用时要注意 Unity 版本要在 5.0 以上，最好使用 Windows 7 以上版本的操作系统，硬件方面需要一个摄像头即可。

4.2.2 导入 OpenCV For Unity 插件包

首先，在 Unity Asset Store 中搜索 OpenCV For Unity，下载完成后导入到 Unity 中。

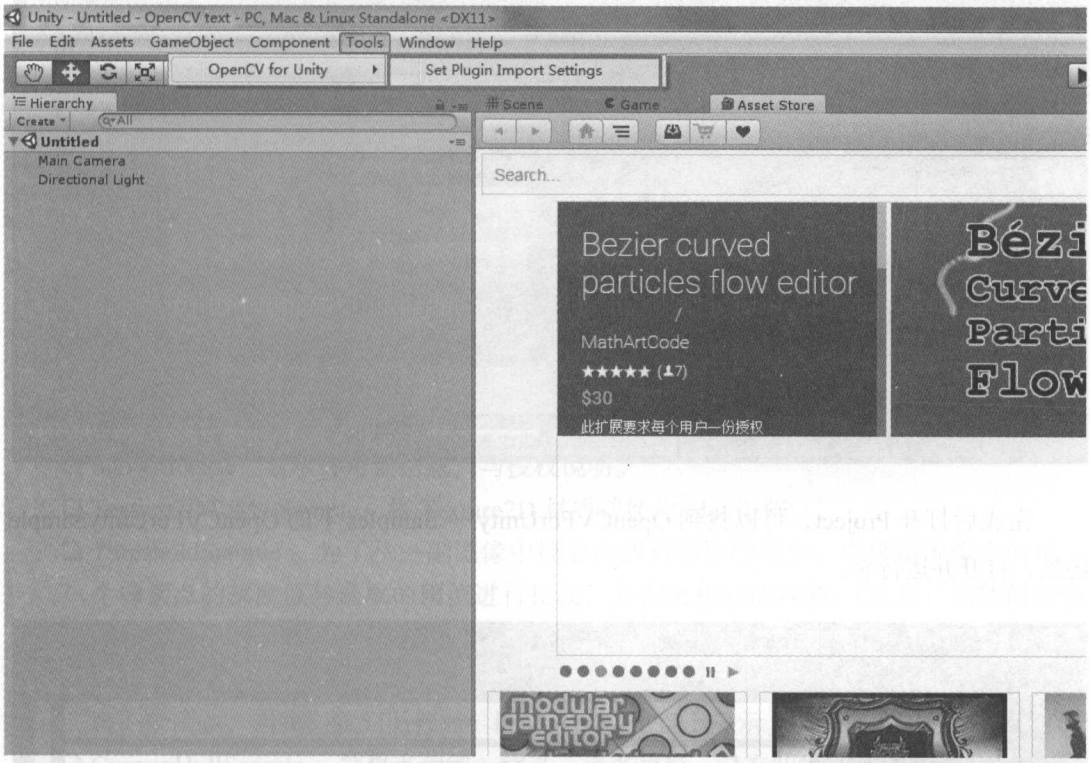


导入完成后，在 Project 中找到 OpenCVForUnity 文件夹将 StreamingAssets 文件夹拖入 Assets 文件夹。



4.2.3　配置 OpenCV For Unity 插件

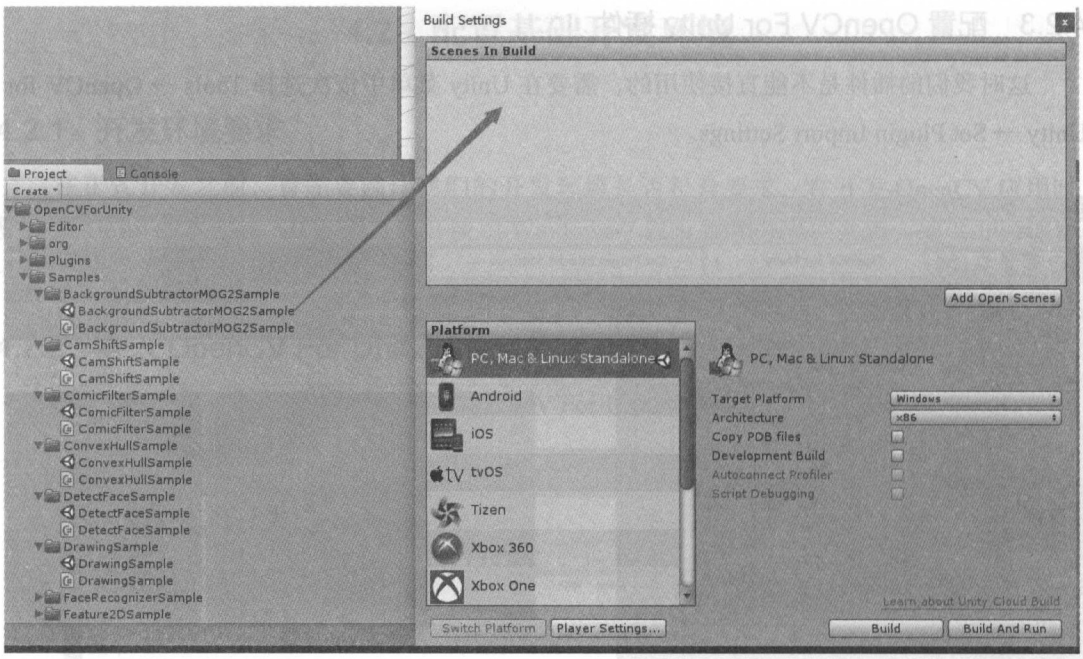
这时我们的插件是不能直接使用的，需要在 Unity 菜单中依次选择 Tools → OpenCV for Unity → Set Plugin Import Settings。



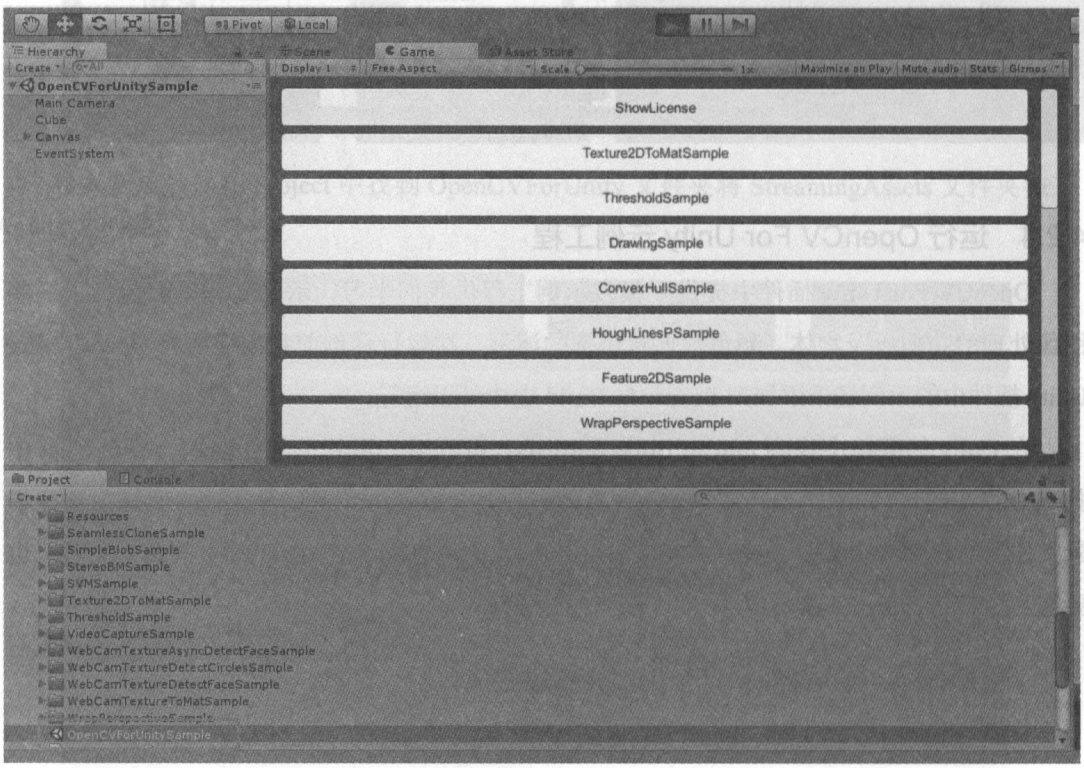
4.2.4　运行 OpenCV For Unity 示例工程

OpenCV For Unity 插件中提供了很多示例工程供我们学习，示例工程里面包括了一些图像处理的功能以及物体、颜色、形状识别的演示。在运行示例前我们需要将 OpenCV For Unity 插件中的示例场景添加到 Scenes In Build 中才可以运行。

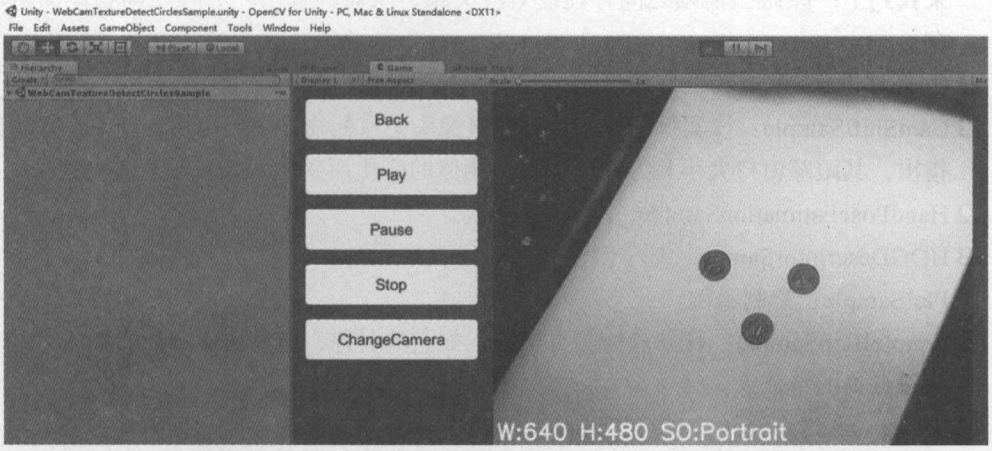
在 Unity 的菜单中找到 File → Build Settings，打开后，不要关闭此窗口，到 Project 中找到 OpenCVForUnity → Samples 文件夹下的 Unity 场景文件，我们需要做的就是把这些场景文件全部拖到 Build Settings 中的 Scenes In Build 窗口中。包括 Samples 下的场景文件，也要拖到 Scenes In Build 窗口中。



完成后打开 Project，可以找到 OpenCVForUnity → Samples 下的 OpenCVForUnitySample 场景，打开并运行它。



这里演示识别圆形示例功能，在列表中找到 WebCamTextureDetectCirclesSample 点击后将打开摄像头，摄像头获取的画面中圆的物体将被标注出来。



这里简单介绍一下示例的功能：

- ShowLicense：显示许可证信息，与授权说明。
- Texture2DToMatSample：将 Texture2D 材质转换为 Mat 矩阵。
- ThresholdSample ：为了从一副图像中提取出我们需要的部分，应该用图像中的每一个像素点的灰度值与选取的阈值进行比较，并作出相应的判断。(注意：阈值的选取依赖于具体的问题。即物体在不同的图像中有可能会有不同的灰度值。)
- DrawingSample ：在输出图像时在图像上添加我们的元素，比如在图像的某一个位置画一个圆圈，当我们看到图像时，圆圈就会在图像上显示。
- ConvexHullSample ：简单点理解，就是一个多边形，没有凹的地方。凸包 (凸壳) 能包含点集中所有的点，凸包检测常应用在物体识别、手势识别及边界检测等领域。
- HoughLinesPSample：检测到的边缘使用 Hough 变换检测直线，可以检出一些多边形物体。
- Feature2DSample ：在特征点附近随机选取若干点对，将这些点对的灰度值的大小组合成一个二进制串，并将这个二进制串作为该特征点的特征描述子，然后进行对比，在两幅图中找到相同特征点，之后画线对点。
- WrapPerspectiveSample：从四对对应点计算透视变换。
- FaceRecognizerSample ：OpenCV 中所有人脸识别的模型都继承自 FaceRecognizer 基类，这个类提供了人脸识别算法的统一接口。
- DetectFaceSample：面部检测示例。
- WebCamTextureToMatSample：在 Unity 中获取摄像头视频流，并转换为 Mat 矩阵。
- WebCamTextureDetectFaceSample ：在 Unity 中获取摄像头视频流，检测面部并在图像中动态标注。
- WebCamTextureAsyncDetectFaceSample：多线程异步处理图像数据检测面部，并在图像中动态标注。

- OpticalFlowSample：光流是由于场景中前景目标本身的移动、相机的运动或者两者的共同运动所产生的；利用图像序列中像素在时间域上的变化以及相邻帧之间的相关性来找到上一帧跟当前帧之间存在的对应关系，从而计算出相邻帧之间物体的运动信息。
- ComicFilterSample：图像输出时被转换为漫画风格。
- CamShiftSample：该实例的作用是跟踪摄像头中目标物体，目标物体初始位置用鼠标指出，其跟踪窗口大小和方向随着目标物体的变化而变化。
- HandPoseEstimationSample: 手势识别示例。
- HOGDescriptorSample: 基于颜色的多对象跟踪示例。
- PlotSample：绘制示例。
- SimpleBlobSample：特征点检测方法，正如它的名称，该算法使用最简单的方式来检测斑点类的特征点。
- BackgroundSubtractorMOG2Sample：高斯处理视频。跟踪运动做前背景分割。
- VideoCaptureSample：视频捕获示例。
- MatchTemplateSample：自定义模板匹配示例。
- StereoBMSample：立体匹配主要是通过找出每对图像间的对应关系，根据三角测量原理，得到视差图；在获取视差信息后，根据投影模型很容易地可以得到原始图像的深度信息和三维信息。
- SeamlessCloneSample：将原图像镶嵌到另外一个目标图像中，并且利用边缘模糊的算法将原图像更好地融合进目标图像里。
- WebCamTextureDetectCirclesSample：使用霍夫变换圆检测算法识别出圆形。
- SVMSample：支持向量机 (SVM) 是一个类分类器，正式的定义是一个能够将不同类样本在样本空间分隔的超平面。 换句话说，给定一些标记 (label) 好的训练样本 (监督式学习) 后，可以使用 SVM 算法输出一个最优化的分隔超平面。
- HOGDescriptor：识别视频流中的动态物体，并标注。

4.3 面 部 识 别

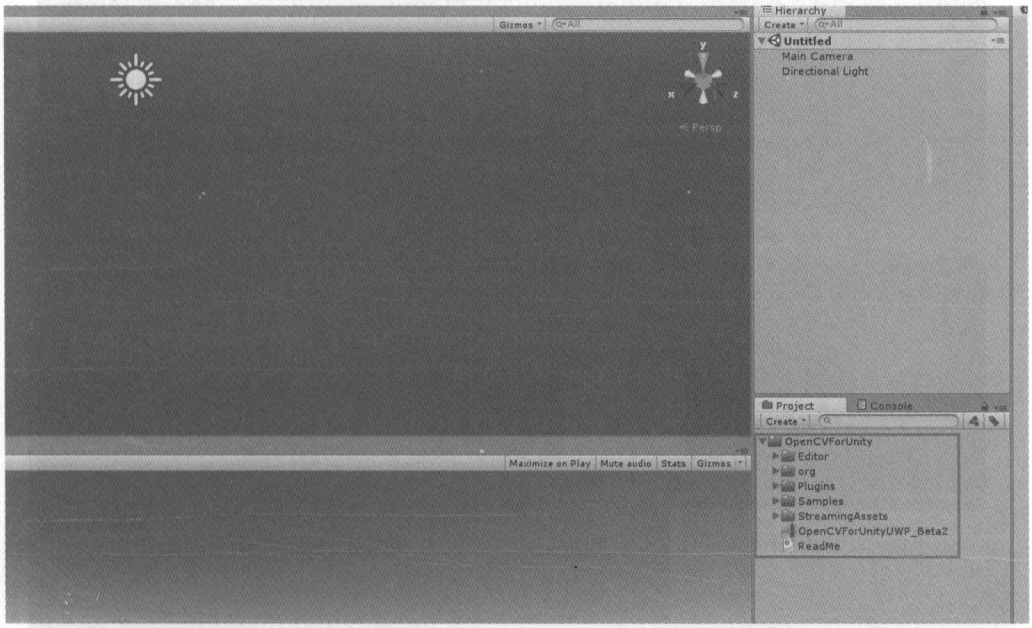
4.3.1 FaceTrackerSample 扩展插件简介

FaceTrackerSample 插件是基于 OpenCV For Unity 插件之上开发的一款人脸识别的插件，主要实现了人脸跟踪、五官识别以及简单表情识别功能。这款插件是免费的，本节主要通过这款插件实现实时面部识别并标注。

4.3.2 场景搭建

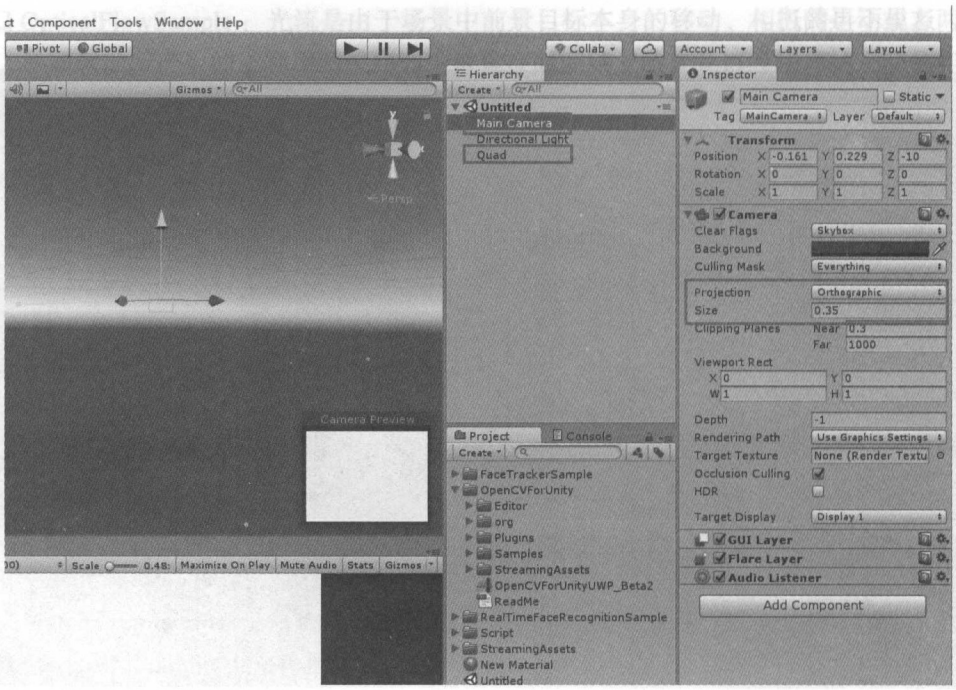
首先需要创建一个 Unity 工程，然后先导入 OpenCV For Unity 插件并配置，在 4.2 节已

介绍，这里不再赘述。

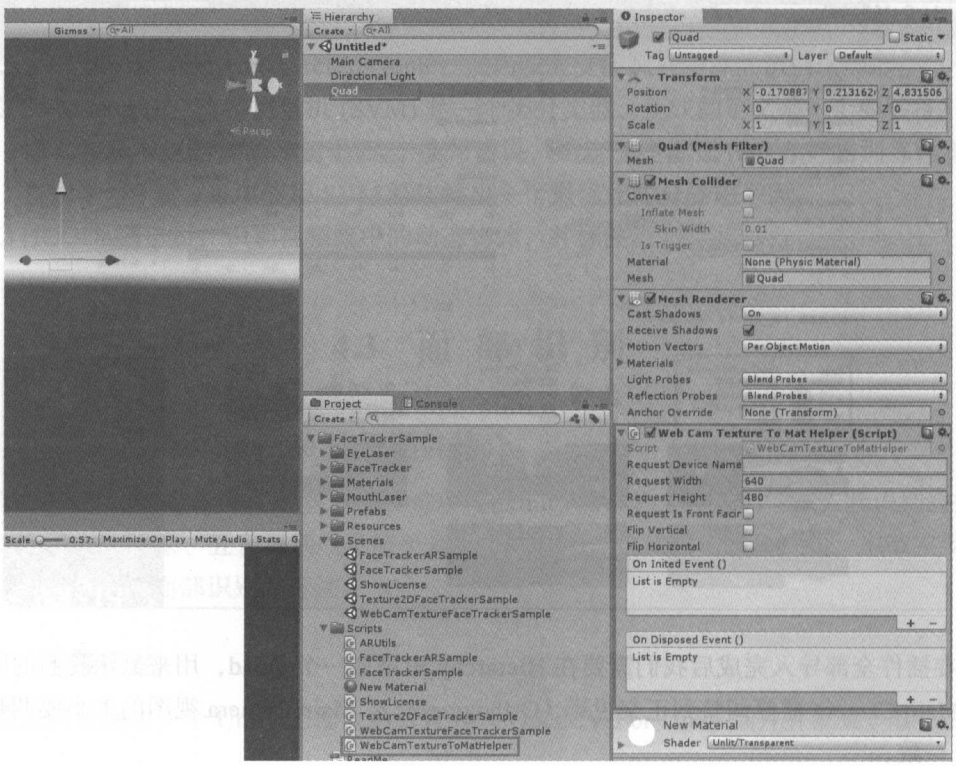


接下来，需要在 Asset Store 中查找 FaceTrackerSample 插件并将其导入工程中。

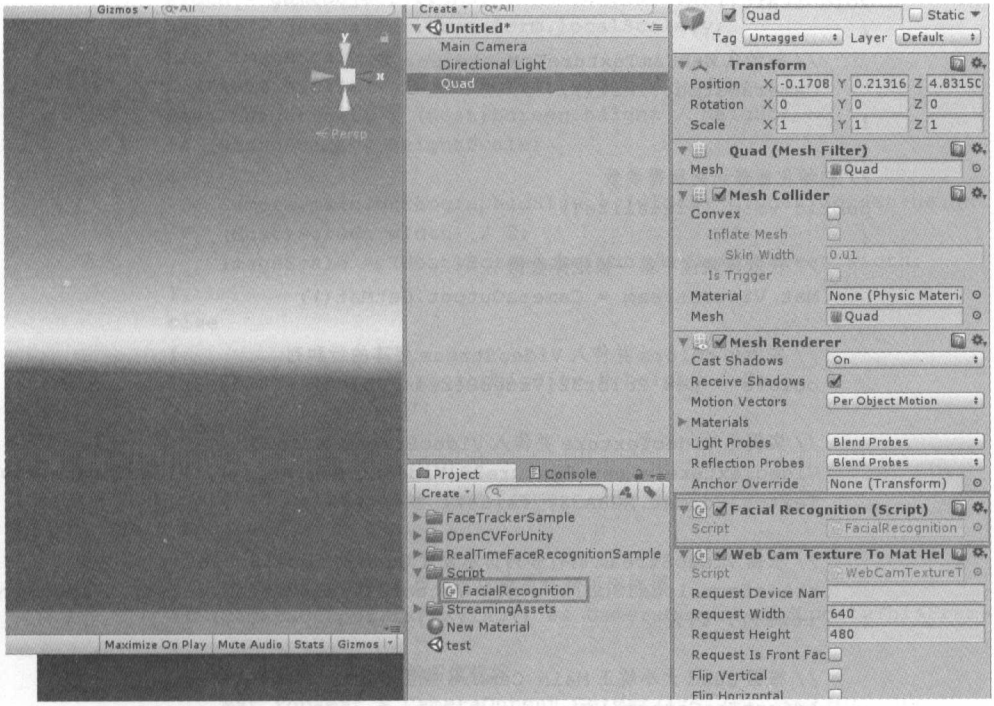
在插件全部导入完成后我们需要在 Hierarchy 中添加一个 Quad，用来显示我们的图像结果。Main Camera 需要调整到正交投影（Orthographic），Main Camera 视图的大小要调整得和 *Quad* 一致。



我们需要在 Project 视图中打开 FaceTrackerSample → Scripts 找到 WebCamTextureToMatHelper 脚本，将它挂载到 Quad 的 Inspector 面板中。



在 Project 视图中新建一个文件夹，并命名为 Script，作为我们存放脚本的文件夹，在文件夹下新建一个脚本，命名为 FacialRecognition，完成后我们需要将 FacialRecognition 脚本挂载到 Quad 的 Inspector 中。



4.3.3　编写面部识别脚本

接下来我们开始编写 FacialRecognition 脚本。FacialRecognition 脚本是实现面部识别功能的主要脚本，功能包括视频流与 Mat 矩阵的转换，面部特征匹配已经在 Mat 矩阵中标注人脸位置输出视频流等功能，脚本中会一一注释讲解，以下是脚本内容。

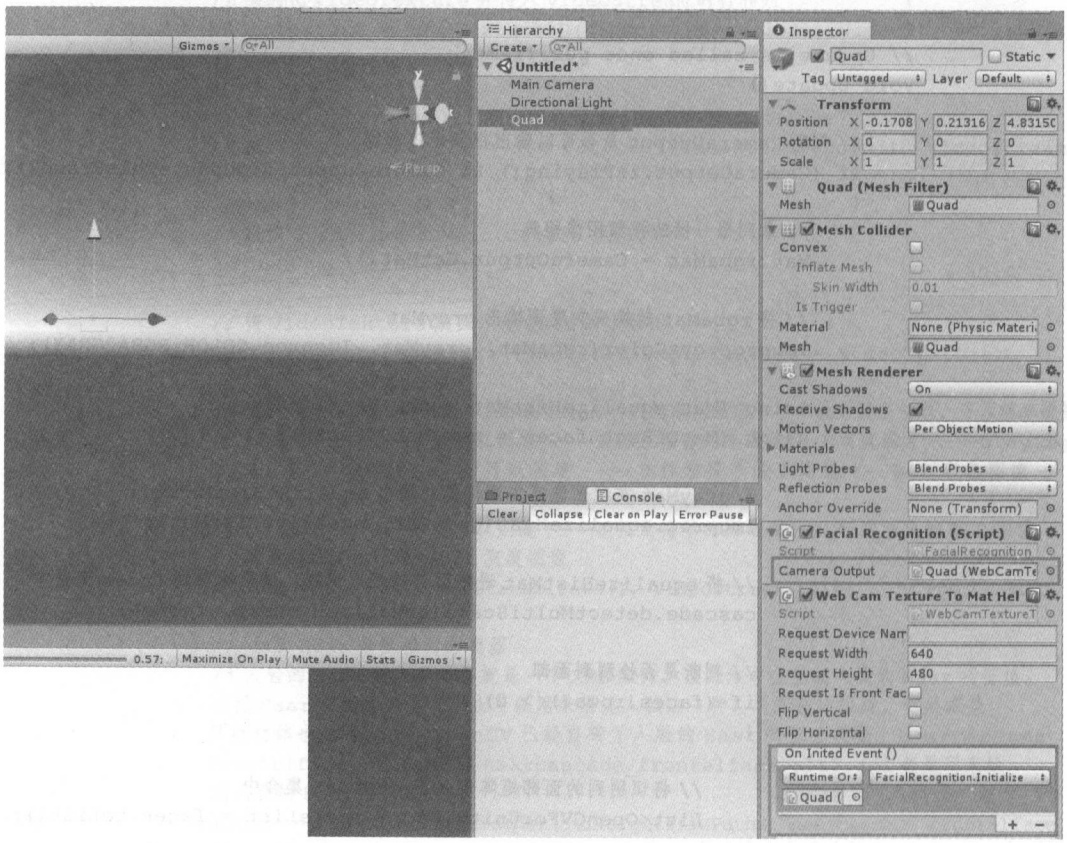
```
using OpenCVFaceTracker;
using OpenCVForUnity;
using OpenCVForUnitySample;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
namespace FaceTrackerSample
{
    public class FacialRecognition : MonoBehaviour
    {
        Texture2D VideoTexture;

        public WebCamTextureToMatHelper CameraOutput;

        Color32[] colors;
```

```
Mat grayMat;

CascadeClassifier cascade;

void Start()
{
    //初始化 WebCamTextureToMatHelper 类
    CameraOutput.Init();
}

//初始化面部识别所需参数
public void Initialize()
{
    //获取视频流中第一帧矩阵数据
    Mat VideoStream = CameraOutput.GetMat();

    //实例化 colors 并传入 VideoStream 矩阵的行和列
    colors = new Color32[VideoStream.cols() * VideoStream.rows()];

    //实例化 VideoTexture 并传入 VideoStream 的行和列
    VideoTexture = new Texture2D(VideoStream.cols(), VideoStream.rows(),
    TextureFormat.RGBA32, false);

    //根据 VideoStream 的行和列信息设置 Quad 的 localScale
    transform.localScale = new Vector3(VideoStream.cols(), VideoStream.
    rows(), 1);

    //根据 Quad 大小矫正 Main Camera 视图参数
    AdjustTheCamera();

    //将 VideoTexture 贴图作为 Quad 的主贴图
    GetComponent<Renderer>().material.mainTexture = VideoTexture;

    //实例化 grayMat 并传入 VideoStream 行列信息，最后一个参数为通道信息，下面简单讲解
    //【1】CV_8UC1--- 则可以创建 ----8 位无符号的单通道 --- 灰度图片 ----grayImg
    //【2】CV_8UC3--- 则可以创建 ----8 位无符号的三通道 ---RGB 彩色图像 ---
    colorImg
    //【3】CV_8UC4--- 则可以创建 ----8 位无符号的四通道 --- 带透明色的 RGB 图像
    //这里我们使用 CV_8UC1 灰度图像
    grayMat = new Mat(VideoStream.rows(), VideoStream.cols(), CvType.CV_8UC1);

    // 加载 Haar 特征检测分类器
    /* 人脸的 Haar 特征分类器就是一个 XML 文件，该文件中会描述人脸的 Haar 特征值。
    当然 Haar 特征的用途可不止可以用来描述人脸这一种，用来描述眼睛，嘴唇或是
    其他物体也是可以的。OpenCV 已经自带了人脸的 Haar 特征分类器。haarcascade_
    frontalface_alt.xml 与 haarcascade_frontalface_alt2.xml 都是用来检
    测人脸的 Haar 分类器 */
    cascade = new CascadeClassifier(Utils.getFilePath("haarcascade_frontalface_
    alt.xml"));
}

    //根据 Quad 大小矫正 Main Camera 视图参数
```

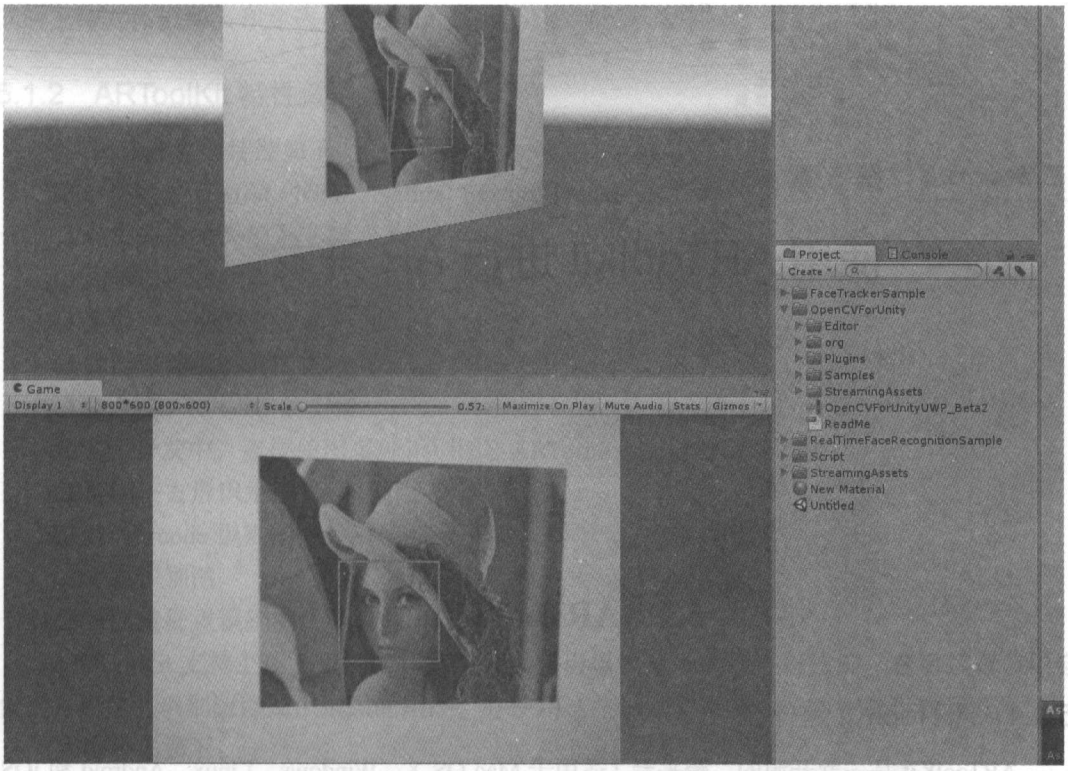
```
void AdjustTheCamera()
{
    float width = 0;
    float height = 0;
    width = gameObject.transform.localScale.x;
    height = gameObject.transform.localScale.y;
    float imageScale = 1.0f;
    float widthScale = (float)Screen.width / width;
    float heightScale = (float)Screen.height / height;
    if (widthScale < heightScale)
    {
        Camera.main.orthographicSize = (width * (float)Screen.height /
        (float)Screen.width) / 2;
        imageScale = (float)Screen.height / (float)Screen.width;
    }
    else
    {
        Camera.main.orthographicSize = height / 2;
    }
}

// Update is called once per frame
void Update()
{
    //判断CameraOutput是否可以输出视频矩阵数据
    if (CameraOutput.isPlaying() && CameraOutput.didUpdateThisFrame())
    {
        //拿到每一帧的视频图像矩阵
        Mat rgbaMat = CameraOutput.GetMat();

        //将rgbaMat转换成灰度图输出grayMat
        Imgproc.cvtColor(rgbaMat, grayMat, Imgproc.COLOR_RGBA2GRAY);

        using (Mat equalizeHistMat = new Mat())
        using (MatOfRect faces = new MatOfRect())
        {
            //grayMat灰度图像直方图均衡化输出equalizeHistMat
            Imgproc.equalizeHist(grayMat, equalizeHistMat);

            //将equalizeHistMat进行面部特征点识别，结果输出到faces变量中
            cascade.detectMultiScale(equalizeHistMat, faces);

            //判断是否检测到面部
            if (faces.rows() > 0)
            {

                //将识别到的面部矩阵传入rectsList集合中
                List<OpenCVForUnity.Rect> rectsList = faces.toList();

                for (int i = 0; i < rectsList.Count; i++)
                {
                    //在rgbaMat矩阵中添加矩形标识线，标识出面部所在位置
```

```
                                Imgproc.rectangle(rgbaMat, new Point(rectsList[i].
x, rectsList[i].y), new Point(rectsList[i].x + rectsList[i].width, rectsList[i].y +
rectsList[i].height), new Scalar(255, 0, 0, 255), 2);
                            }
                        }
                    }

                    // 将标识过的矩阵转换为 Texture2D 材质
                    Utils.matToTexture2D(rgbaMat, VideoTexture, colors);
                }
            }
        }
    }
```

脚本编写完成后在 Quad 的 Inspector 面板中找到 FacialRecognition 脚本下的 Camera Output，将 Hierarchy 面板中的 Quad 拖入。然后在 Quad 的 Inspector 面板找到 On Inited Event 添加一个方法，之后将 Hierarchy 面部中的 Quad 拖入，选择 FacialRecognition 中的 Initialize 方法。



连接好网络摄像头，开始运行工程，之后就可以在 Game 窗口中看到摄像头的画面，如果画面中有面部出现会被红色的矩形线框选出来。


Project
Console
Create
FaceTrackerSample
OpenCVForUnity
Editor
org
Plugins
Samples
StreamingAssets
OpenCVForUnityUWP_Beta2
ReadMe
RealTimeFaceRecognitionSample
Script
StreamingAssets
New Material
Untitled
Game
Display 1
800*600 (800x600)
Scale
0.57:
Maximize On Play
Mute Audio
Stats
Gizmos

Chapter 3 第 5 章

使用 ARToolkit 进行 AR 开发

5.1 ARToolKit 简介

5.1.1 ARToolKit 是什么

ARToolKit 是一款开源的，跨平台（适用于 Mac OS X、Windows、Linux、Android 和 iOS）AR 开发 SDK，使用 ARToolKit 可以大幅简化 AR 开发。可以将它应用于游戏、视觉、科研的开发。ARToolKit for Unity 是 Unity3D 的插件版本，可以和 Unity3D 无缝开发跨平台的 AR 应用。

ARToolKit for Unity 的工作方式

ARToolKit for Unity 的工作流为：

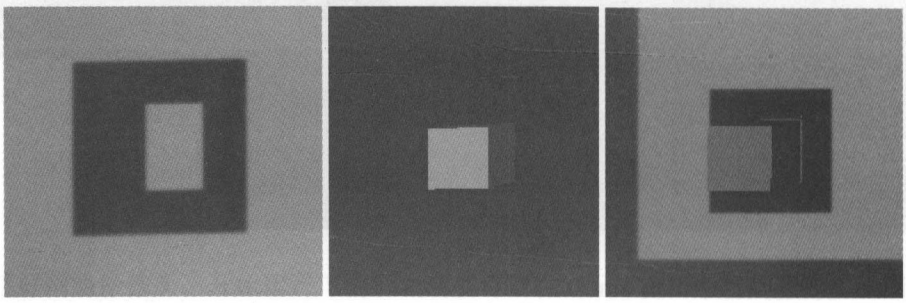
1）获取实际的摄像头的输入源图像（左图）。

2）寻找输入源图像中的识别图。

3）获取识别图在图中的 Transform 数据。

4）将与识别图对应的对象的 Transform 与标识图像的同步。

5）使用 Unity 渲染后将结果（中图）与摄像头的输入结果混合后输出（右图）。



插件本身处理了 Unity 与摄像头之间的通信，并将摄像头的影响作为了一个视频背景，

同时支持立体摄像头和穿透（半透明）显示。

5.1.2 ARToolKit 特性简介

ARToolkit 主要特性如下：

- 自然特性点追踪（NFT：Natural Feature Tracking）
- 强大的摄像头校正支持
- 同步追踪和立体摄像机的支持
- 多编程语言的支持
- 针对移动设备的优化
- Unity3D 和 OSG 的支持

在实际开发中，需要使用多种识别图。ARToolKit 支持的识别图包括：

- 传统正方形识别图
- 2D barcode 识别图
- 多重识别图
- 自然特征点识别图（自然图片）

ARToolKit 支持上述识别图的任意组合，并可以快速准确地进行追踪。在后续章节中会依次介绍这些识别图以及识别图的用法，并使用这些识别图在 Unity3D 中进行项目开发。

系统需求如下：

- 一个支持 ARToolKit 的摄像头 (或 webcam）
- Unity3D 5.0 以上的版本
- 如果在 Android 上部署，则需要安装 Android Build support for Unity
- 如果在 iOS 上部署，则需要安装 iOS Build support for Unity 和 XCode 4.2+

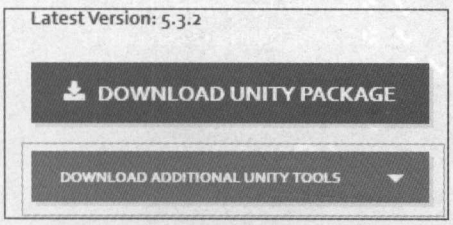
重要提示：

- 无法在 Windows 上创建 iOS 或 Mac OS X 的 App
- 无法在 Mac OS X 上创建 Windows Store、Windows Phone 和 Windows Universal Platform 的 App

5.1.3 ARToolKit 插件包导入

在官网获取插件

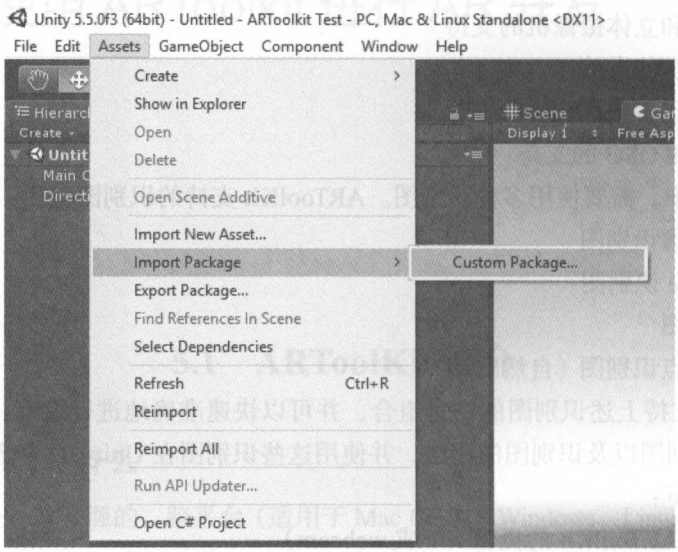
网址为 http://artoolkit.org/download-artoolkit-sdk。

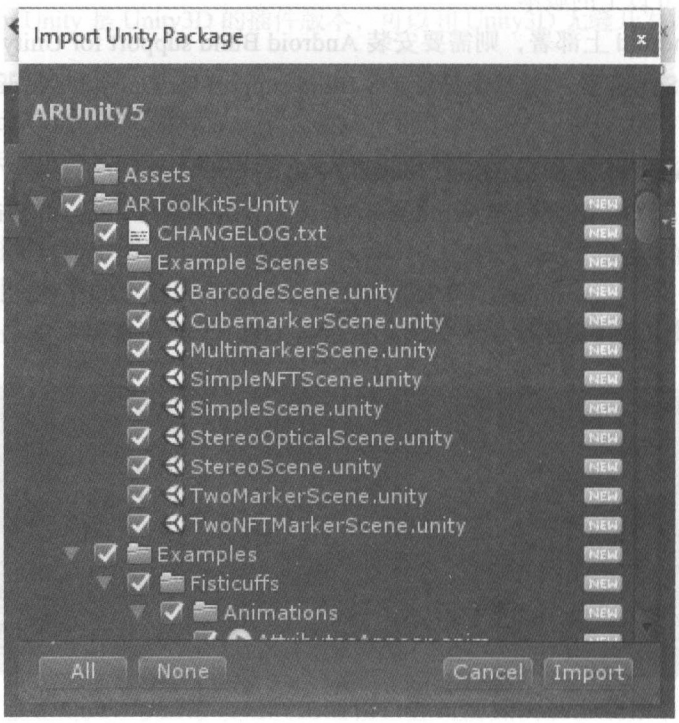


下载 Additional Unity Tools，其中包含 Unity 插件和脱机工具。将下载的压缩包解压至 E 盘（后文中用到 Unity Tools 的地方会使用这个路径，但解压位置可以自由选择），并重命名为 ARToolKit。

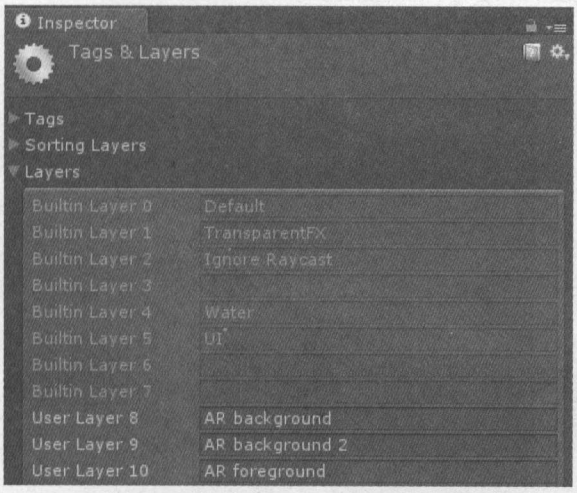
将插件导入 Unity

在菜单栏中依次选择 Assets → Import Package → Custom Package，导入下载的插件。





ARToolKit 导入后会建立 3 个层：AR background、AR background 2 和 AR foreground。



这些层主要是用来区分哪些是 ARToolKit 渲染的物体，方便在实际应用中混合 Camera 的渲染结果。

5.1.4 ARToolKit 中的目录简介

ARToolKit 中的目录主要包括：

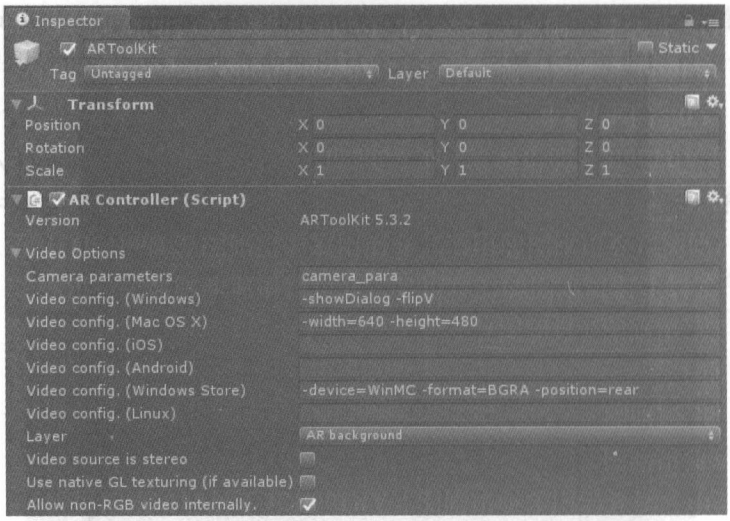
- ARToolKit5-Unity(插件的主目录)
- Example Scenes(简单的例子场景)
- Examples(示例项目)
- Materials(例子场景中用到的材质)
- Resources(摄像机配置文件，标识的特征文件和视觉文件以及 Shader)
- Scripts(插件代码和编辑器代码，开发者主要需要的类和函数)
- Plugins(各个平台的运行库)
- StreamingAssets(多重标识的配置文件，NFT 的特征文件)

5.2 搭建一个简单的 AR 场景

搭建一个基础 ARToolKit 场景不需要写任何代码，插件代码可以直接创建一个 AR 对象。然而对于比较灵活的功能需求来说，使用脚本可以完全控制所有 ARToolKit 中的功能。包括动态添加识别图，开始 / 停止追踪，更新参数等。

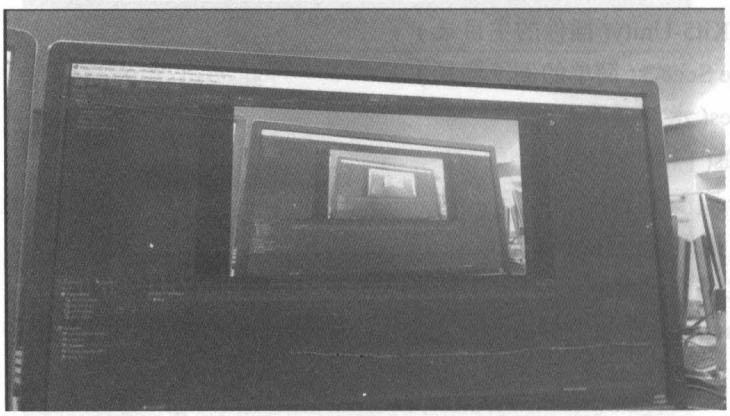
5.2.1 创建并设置 AR Controller

创建一个空的 Game Object，重命名为“ARToolKit”，为其添加 AR Controller 组件。取消勾选 Use native GL texturing(if available)，选中 Allow non-RBG video internally。



现在运行场景，摄像头连接驱动正确的话，在游戏画面中是可以看到摄像头的图像的。

注意 某些摄像头在调试中开启对话框可能会导致 Unity 程序崩溃，只要将 Video config. (Windows) 中的 -showDialog 删除即可。

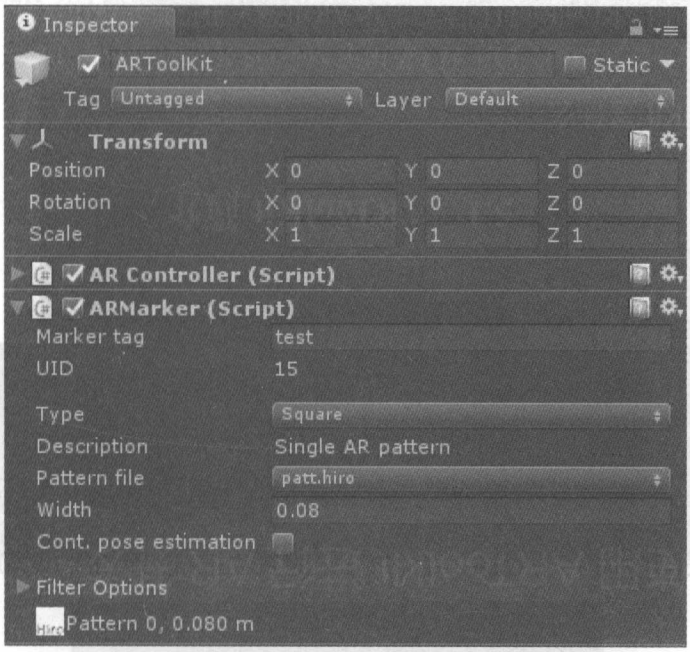


在 AR Controller 中有一个重要的选项 Content Mode，它有以下三个选项：

- Stretch：无视屏幕比例，拉伸至全屏（不建议，会导致变形）
- Fit：以摄像头输入尺寸拉伸至屏幕（宽度和高度中的较小值），显示屏幕比例与输入图像比例不符时会出现未填充区域（通常为黑边）
- Fill：以摄像头输入尺寸拉伸至屏幕（宽度和高度中的较大值），显示屏幕比例与输入图像比例不符时会裁剪部分边缘输入图像
- One To One：以 1 ：1 的实际像素显示输入图像，输入像素和屏幕像素不同时会发生输入像素剪裁或出现未填充区域

5.2.2　创建并设置 ARMarker

在刚才创建好的 Game Object 上添加 ARMarker 组件，并将 Marker tag 设为“test”。



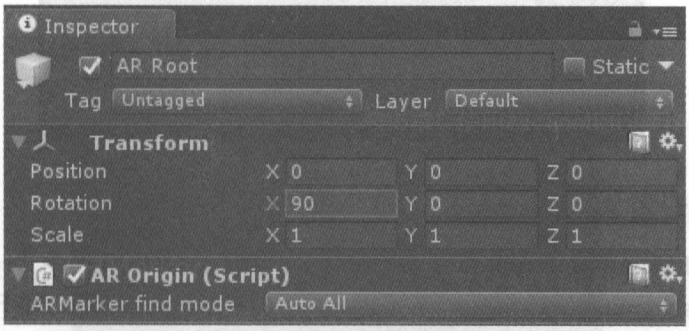
当 ARMarker 的 Type 为 Square 时（如上图所示），ARMarker 的编辑器代码会自动定位“Resources/ardata/markers”文件夹下创建的标识特征文件，并在 Pattern file 的下拉菜单中添加检测到的标识特征文件。

ARMarker 的类型及应用在后续的章节中会依次介绍，现在保持这些默认值即可。

5.2.3　创建并设置 AR Origin 和 AR Tracked Object

AR Origin

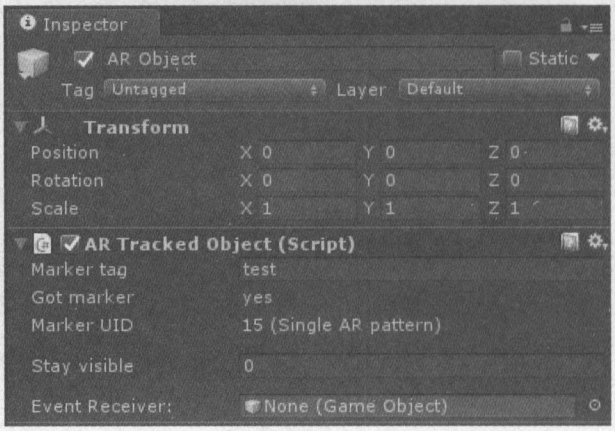
在场景中创建空 Game Object 并命名为 AR Root，为其添加 AR Origin 组件。并将其的 Rotation.x 设置为 90。



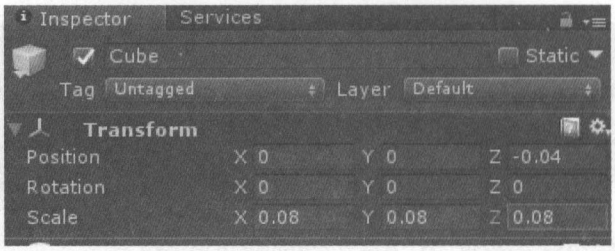
有关 AR 场景的所有物体都应该放置在此对象下。

AR Tracked Object

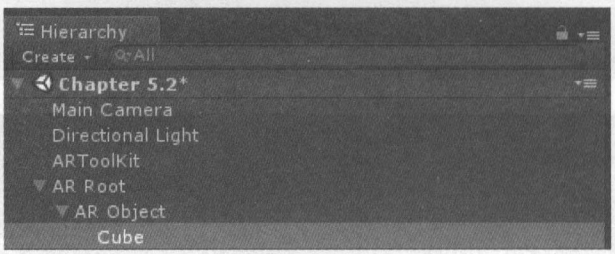
在 AR Root 下创建 Game Object 命名为 AR Object，为其添加 AR Tracked Object 组件，并将 Marker tag 设为“test”（ARMarker 以此为依据追踪该对象）。



在 AR Object 下创建一个 Cube，并将 Cube 的 Position 设为（0，0，-0.04），Scale 设置为（0.08，0.08，0.08）。

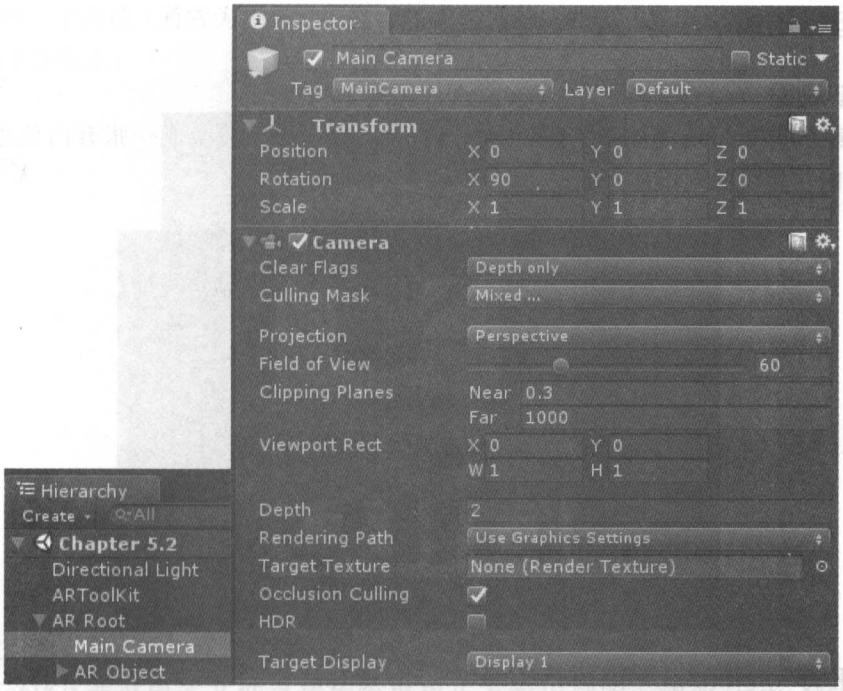


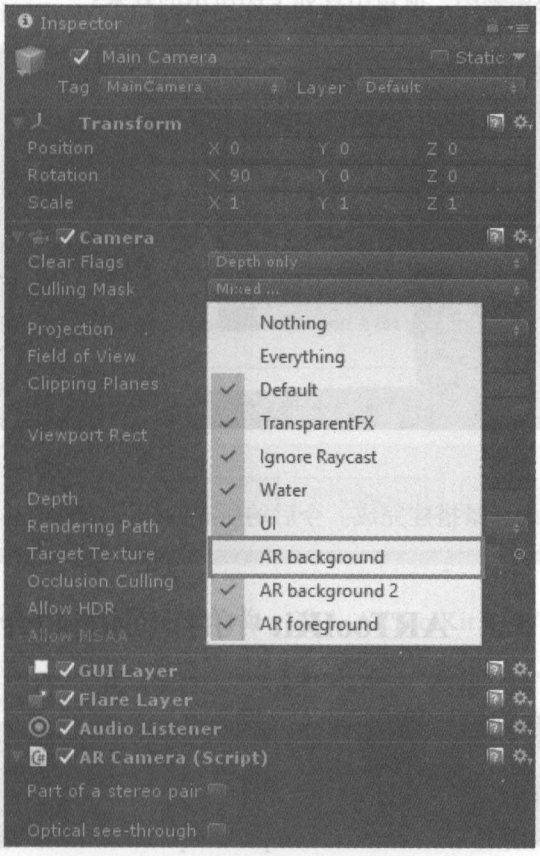
最终的 Hierarchy 如下图所示：



5.2.4 创建并设置 ARCamera

将场景中的 Main Camera 移动到 AR Root 下，将 Clear Flags 设为 Depth only，Culling Mask 去掉 AR Background，将 Depth 设为 2，再添加 ARCamera 组件，如下图所示。


Inspector
Main Camera
Static
Tag MainCamera
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 90 Y 0 Z 0
Scale X 1 Y 1 Z 1
Camera
Clear Flags Depth only
Culling Mask Mixed ...
Projection Perspective
Field of View 60
Clipping Planes Near 0.3
Far 1000
Viewport Rect X 0 Y 0
W 1 H 1
Depth 2
Rendering Path Use Graphics Settings
Target Texture None (Render Texture)
Occlusion Culling
HDR
Target Display Display 1
Hierarchy
Create
Chapter 5.2
Directional Light
ARToolKit
AR Root
Main Camera
AR Object


Inspector
Main Camera
Static
Tag MainCamera
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 90 Y 0 Z 0
Scale X 1 Y 1 Z 1
Camera
Clear Flags Depth only
Culling Mask Mixed ...
Projection
Field of View
Clipping Planes
Viewport Rect
Depth
Rendering Path
Target Texture
Occlusion Culling
Allow HDR
Allow MSAA
Nothing
Everything
Default
TransparentFX
Ignore Raycast
Water
UI
AR background
AR background 2
AR foreground
GUI Layer
Flare Layer
Audio Listener
AR Camera (Script)
Part of a stereo pair
Optical see-through

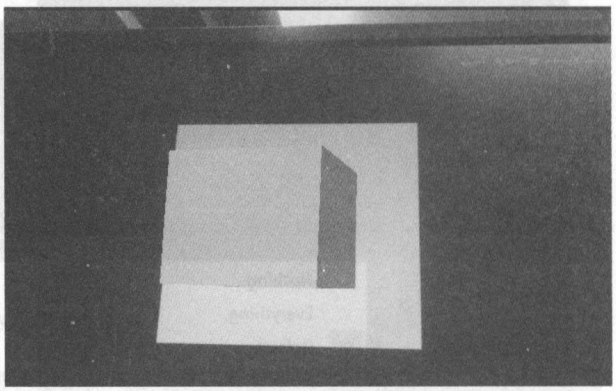
5.2.5 运行场景

准备识别图

将下载的 Windows Tools 的 doc/patterns/Hiro pattern.pdf 转换成一张有白色边框的图片（或者运行时将摄像头对准此页）。

运行程序

将摄像头对准上述识别图，应该出现如下图所示的效果：



至此，ARToolKit 的场景搭建完成。今后的所有的拓展场景均基于此场景。

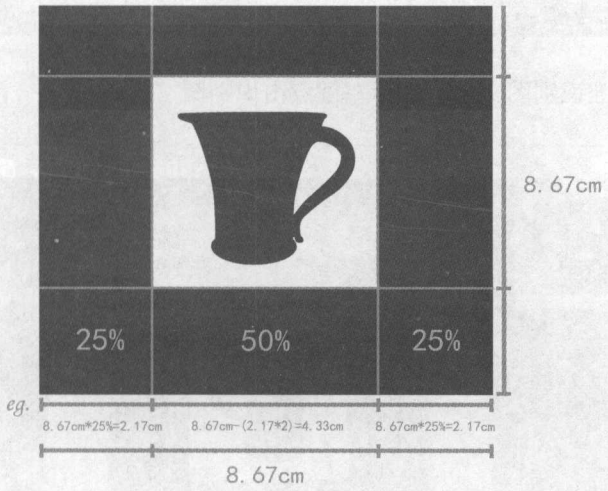
5.3 ARToolKit 中的识别图简介

5.3.1 传统模板正方形识别图

创建自定义识别图

传统模板正方形识别图（Traditional Template Square Marker）通常是由浅色（通常为白

色）包围的一个深色（通常为黑色）正方形黑边包围的一张图（黑边的宽度为正方形边长的25%），如下图所示：

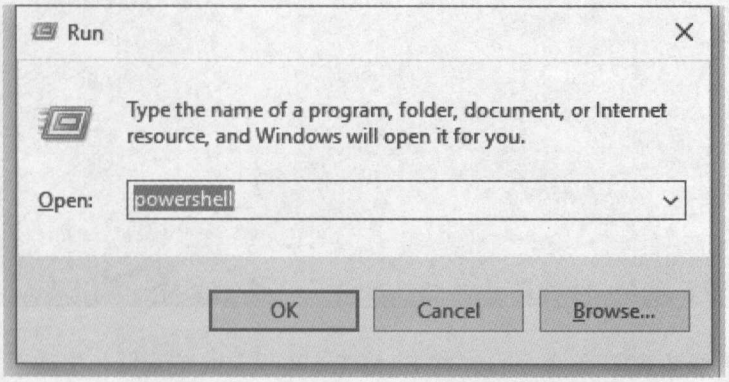


创建自定义识别图的模式文件

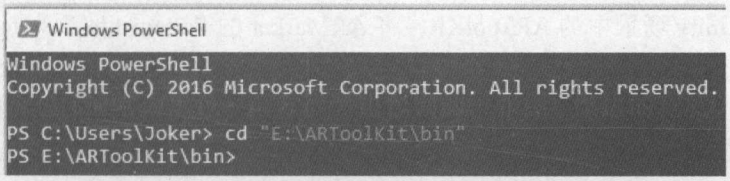
ARToolKit 通过模式文件寻找传统模板正方形识别图，例如 Unity 插件目录下的 Assets\ARToolKit5-Unity\Resources\ardata\markers\patt.hiro.txt。当使用自定义识别图时，需要使用 ARToolKit 提供的 Windows 工具。

1）以上述条件制作识别图。

2）按快捷键 Win+R 打开运行窗口，输入“cmd”或“powershell”打开命令行窗口。



3）执行“cd“E:\ARToolKit\bin”” 命令，进入 ARToolKit 的 Windows 工具的 bin 目录。



4）执行 .\mk_patt.exe，选择默认设置（直接按回车）打开摄像头。

```
PS E:\ARToolKit\bin> .\mk_patt.exe
Enter camera parameter filename (default: 'Data/camera_para.dat'):
Using default video config.
```

5）让摄像头对准识别图文件，鼠标左键点击识别图。

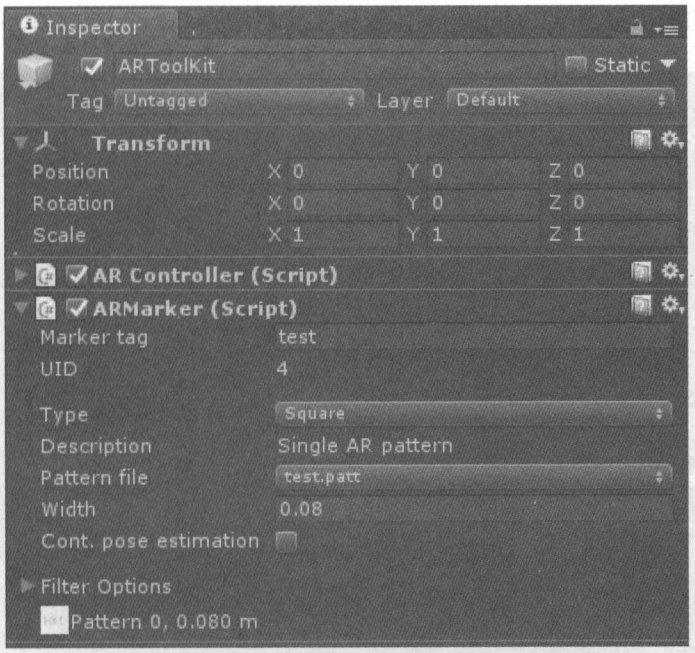
6）输入模式文件的文件名“test.patt.txt“，按回车。

```
PS E:\ARToolKit\bin> .\mk_patt.exe
Enter camera parameter filename (default: 'Data/camera_para.dat'):
Using default video config.
bmpBufferSize = 921600, width = 640, height = 480
Image size (x,y) = (640,480)
---------------------------------------
SIZE = 640, 480
Distortion factor: k1=0.1147807688, k2=-0.5208189487, p1=-0.0002069871, p2=-0.0040593124
                   fx=674.171631, fy=633.898087, x0=318.297791, y0=237.900467, s=0.993923
678.29391 0.00000 318.29779 0.00000
0.00000 637.77411 237.90047 0.00000
0.00000 0.00000 1.00000 0.00000
---------------------------------------
Enter filename: test.patt.txt
  Saved
```

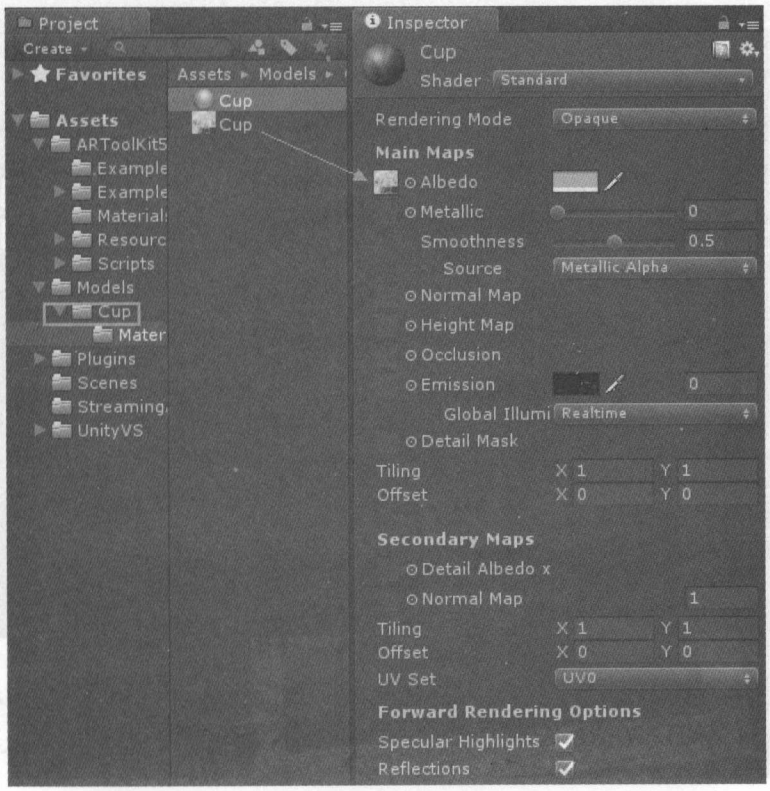
7）关闭摄像头窗口和命令行，将 E:\ARToolKit\bin\test.patt.txt 移动至项目目录的 Assets\ARToolKit5-Unity\Resources\ardata\markers\ 目录下。

在项目中检测识别图

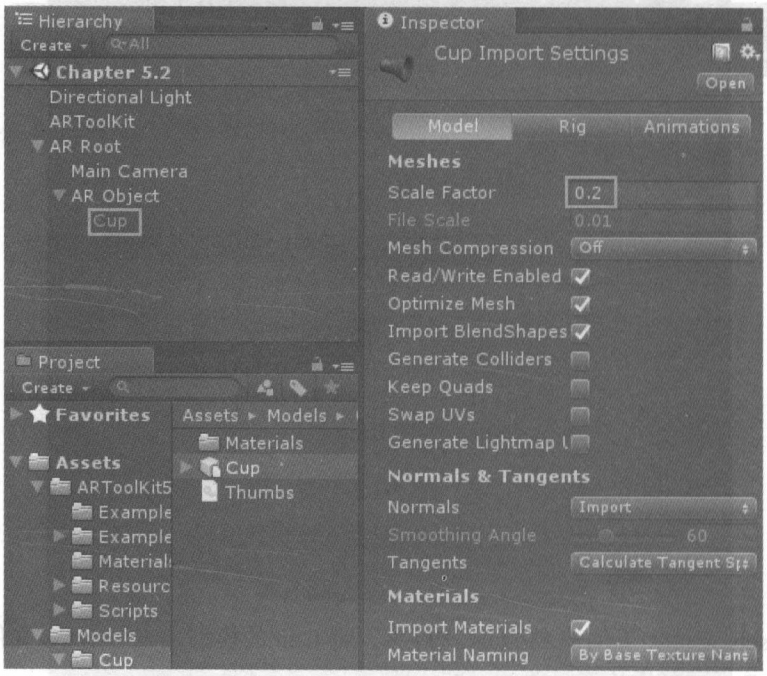
1）选择 Unity 场景中的 ARToolKit，将 ARMarker 的 Pattern file 改为 test.patt（在 Unity 重新获得焦点时，ARToolKit 的 Editor 代码会检测到 Assets\ARToolKit5-Unity\Resources\ardata\markers 目录下新增了模式文件，因此这里的下拉菜单会动态添加刚才移动至该目录的模式文件）。



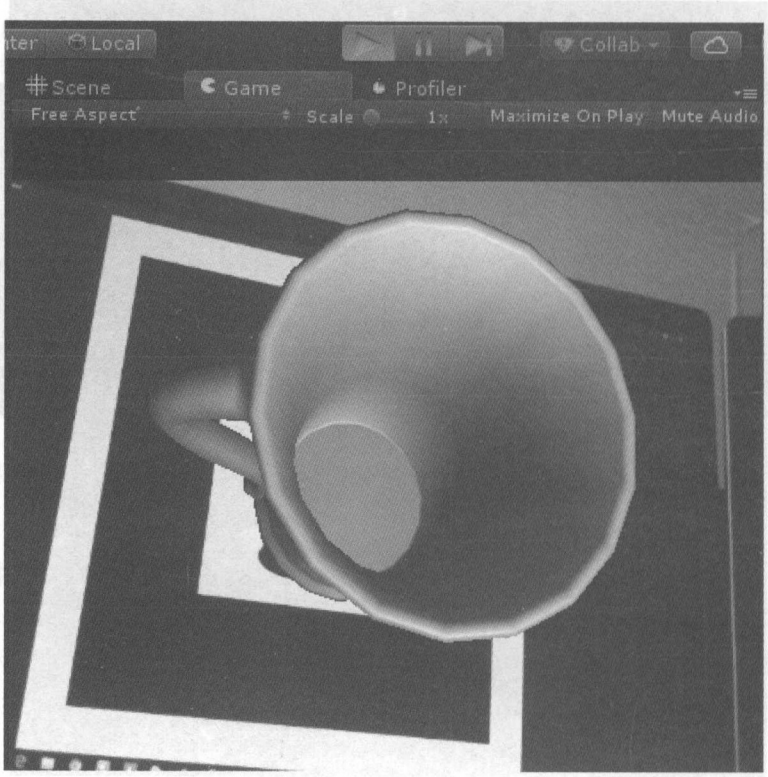
2）在 Assets 文件夹下创建 Models 文件夹，将 Cup 的模型放在 Models 文件夹下，如果材质丢失，将 Cup 的材质拖至材质的对应位置。



3）将 Cup 的模型拖至 AR Object 下，删除之前创建的 Cube，并调 Cup 的大小。



4）运行项目，将摄像头对准识别图，会在识别图位置出现杯子的模型。



5.3.2　2D-Barcode 识别图

2D-Barcode 的特点如下：

2D-Barcode 识别图不是条形码，因为没有准确的中文翻译所以直接使用原英文。

2D-Barcode 是 ARToolKit 内置的一种识别图，位于 ARToolKit 的 Windows 工具目录 E:\ARToolKit\doc\patterns\Matrix code 3x3 (72dpi)。

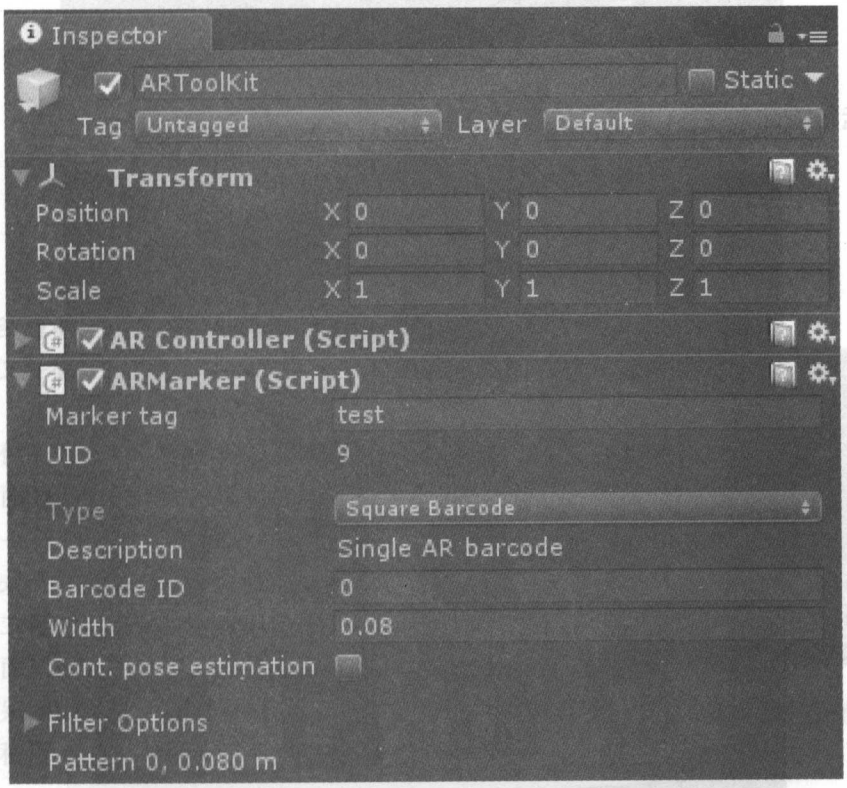
2D-Barcode 的优点如下：

- 运行时性能开销小，可以在多重识别图或识别图数量较多的场景下应用。
- 形状简单且不对称，误认率很低。
- 无需生成模式文件，使用方便。

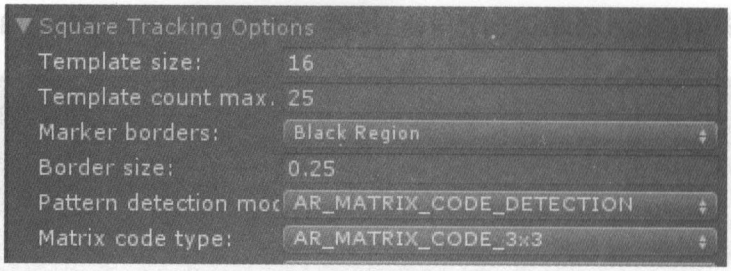
2D-Barcode 的缺点也很明显，无法自定义，识别图数量有限且自由度低。

在项目中使用 2D-Barcode

1）选择 Unity 场景中的 ARToolKit，将 ARMarker 的 Type 改为 Square Barcode。保持 Barcode ID 为 0，这里的 ID 对应 E:\ARToolKit\doc\patterns\Matrix code 3x3 (72dpi) 文件夹下的文件名。



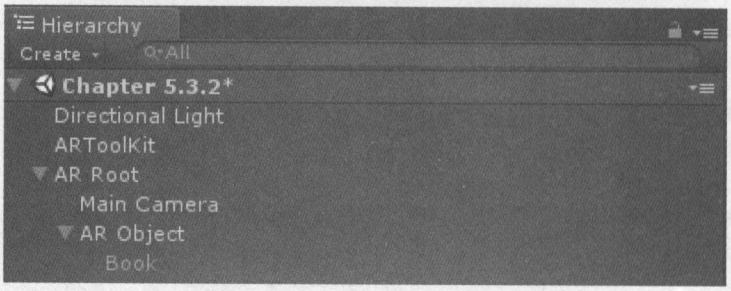
2）将 AR Controller 的 Square Tracking Options 展开，将 Pattern detection mode 改为 AR_MATRIX_CODE_DETECTION，确保 Matrix code type 为 AR_MATRIX_CODE_3x3。



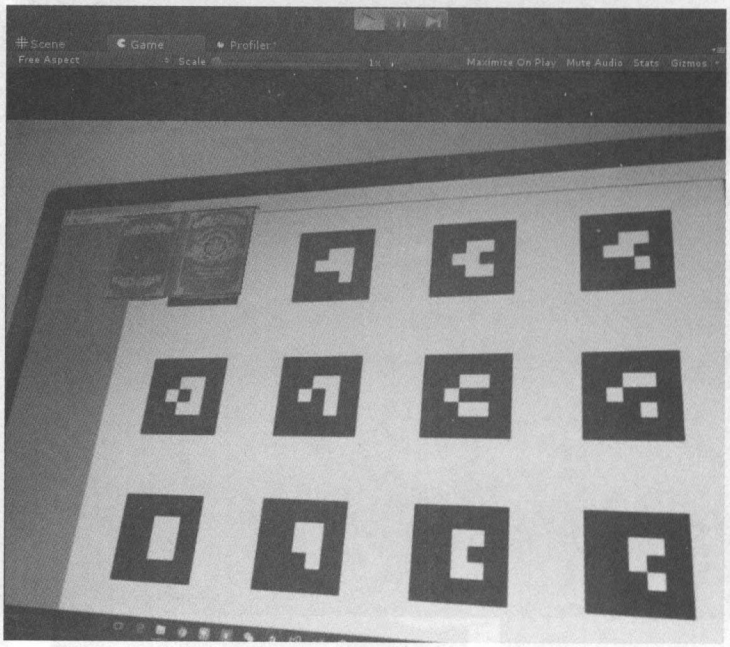
3）打开 E:\ARToolKit\doc\patterns\Multi pattern 4x3 (A4).pdf。

4）将 Book 文件夹（模型文件）复制到 Assets/Models 目录下。

5）将 Book 模型拖至 AR Object 下，并调整其显示大小和位置。



6）运行项目文件，让摄像头对准该文件，会出现 AR Tracked Object。



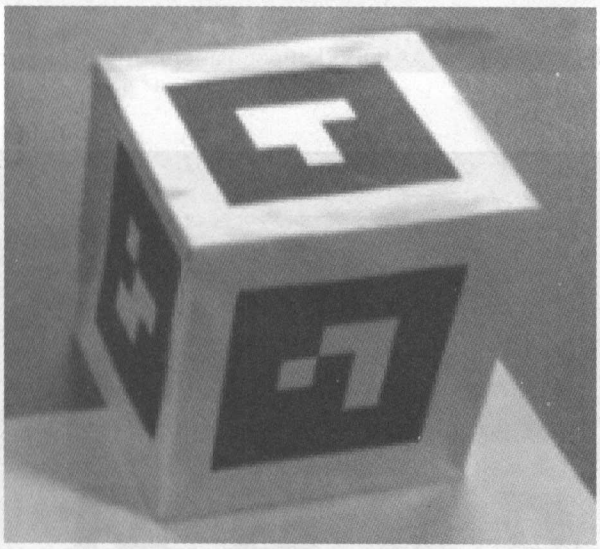
此时如果在 Inspector 面板中切换 ARMarker 的 Barcode ID，AR Tracked Object 的位置会动态地发生变化。

5.3.3　多重识别图

多重识别图并不是 N 个 ARMarker 和 N 个与其对应的 ARTrackedObject，而是由多个识别图构成一个整体的识别图来追踪一个 AR Tracked Object。它具有以下优点：

- 更强大的追踪能力：部分（不超过一半）的子识别图被遮挡时依然可以正确追踪。
- 更准确的定位能力：所有角落的子识别图都用于计算 Transform，使其更准确。
- 可以使用趋势定位：使用统计技术改进误识别引发的识别失败。

多重识别图支持平面识别及立体识别两种，平面识别即将多个识别图打印到一张纸上，立体识别图可以是一个如图所示的立方体。



创建多重识别图的配置文件

配置文件如下所示：

- 以‘#’开头的为注释
- 文件的第一行非注释内容为识别图的数量
- 后面为每个识别图的说明区域
- 第一行为 Barcode ID（如果识别图为 Barcode）或模式文件名（正方形识别图）
- 第二行为识别图打印后的尺寸（单位为毫米）
- 后面三行为相对于原点的变换矩阵，在仅适用平面识别图的情况下，第一行最后一个值为 x 轴的偏移量，第二行最后一个值为 y 轴的偏移量

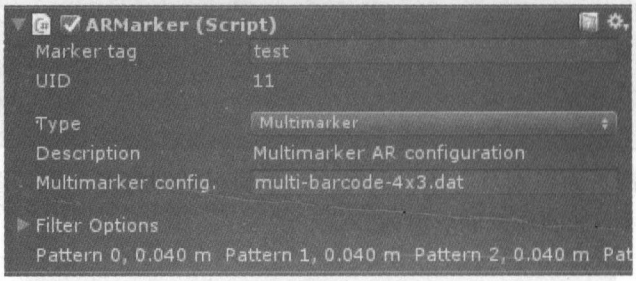
multi-barcode-4x3.dat

```
1  # This is comment
2  12
3
4  00
5  40.0
6  1.0 0.0 0.0 -105.00
7  0.0 1.0 0.0 70
8  0.0 0.0 1.0 0.0
```

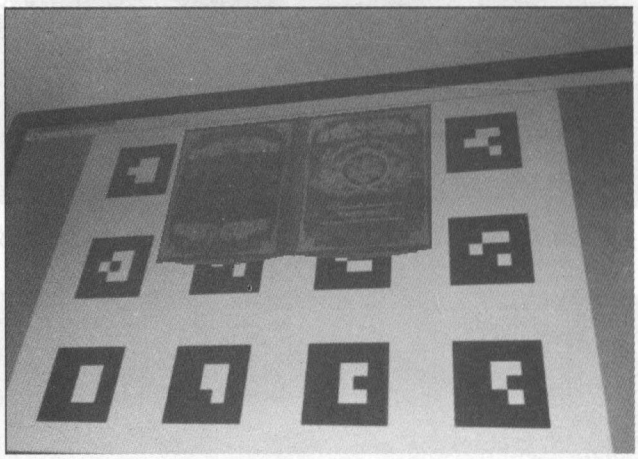
在项目中使用多重识别图

我们直接使用 ARToolKit 提供的配置文件，位于 Unity 项目的 Assets\StreamingAssets\multi-barcode-4x3.dat 目录下。

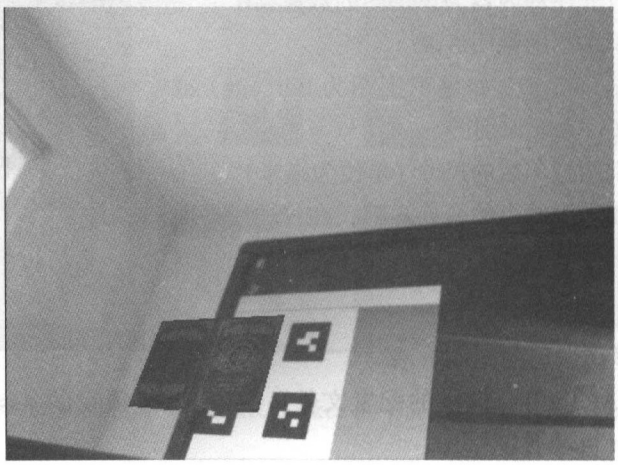
1）选择Unity场景中的ARToolKit，将ARMarker的Type改为Multimaker，在Multimaker config中填入multi-barcode-4x3.dat。



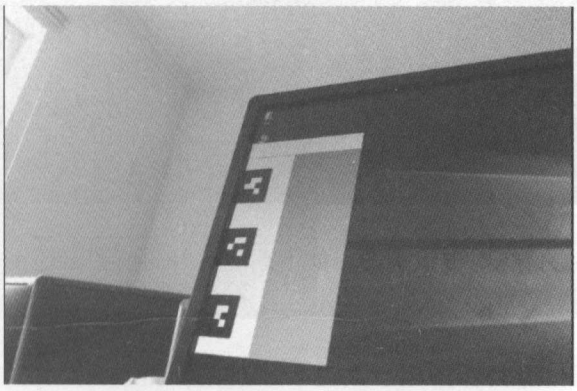
2）运行场景，将摄像机对准E:\ARToolKit\doc\patterns\Multi pattern 4x3 (A4).pdf，出现AR Tracked Object。



3）将文件移动到屏幕边缘，逐渐使子识别图不可见，有6个在屏幕中时仍然可以看到AR Tracked Object。



4）当识别图的数量小于 6 个时，ARTrackedObject 消失，追踪失败。



5.3.4　特征点识别图

特征点识别图即使用任意形状的自然图片作为识别图，可以是照片、卡通图案等。然而，ARToolKit 的识别还是有一些条件限制的。

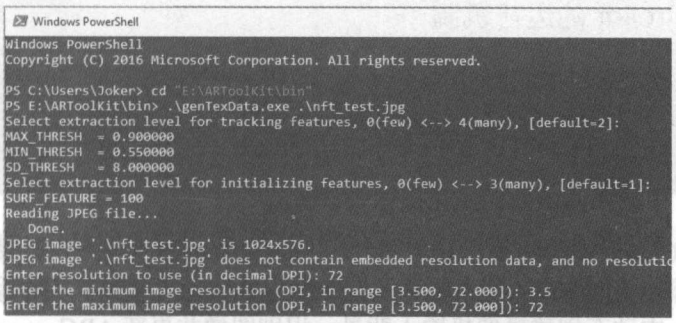
- 提供的图片需要是一张 jpeg 格式的图片，如果是实际图片需要使用扫描仪或相机获取
- 图片本身需要提供一定程度的细节及边缘，完全模糊和纯色图片无法识别
- 图片不能为全对称图形

创建识别图的数据集

1）将识别图 nft_test.jpg 放置于 E:\ARToolKit\bin（ARToolKit 的工具目录）。

2）打开 cmd 或 powershell，进入 E:\ARToolKit\bin。

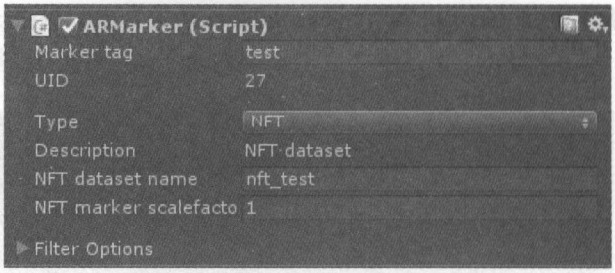
3）执行 .\genTexData.exe .\nft_test.jpg，如下图所示输入数据，等待输出结果。



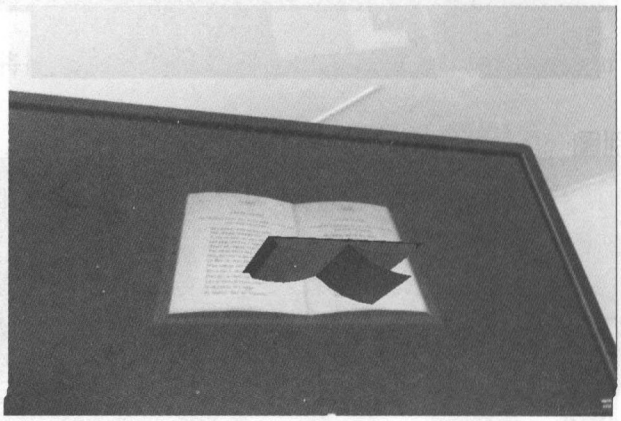
4）进入 E:\ARToolKit\bin 文件夹，将生成的文件 nft_test.fset、nft_test.fset3、nft_test.iset 放置在 Unity 项目文件夹 Assets\StreamingAssets 下。

在项目中使用特征点识别图

1）选择 Unity 场景中的 ARToolKit，将 ARMarker 的 Type 改为 NFT，将 NFT dataset name 改为 nft_test。



2）运行场景，将摄像机对准识别图，如下图所示书的模型出现在书的中间。



5.4 ARToolKit 的进阶内容

5.4.1 AR Controller 的运行机制

ARController 是一个单例（Singleton），管理 ARToolKit 的初始化、设置、运行和停止，并负责创建和管理 AR 追踪，包括实体摄像机的输入处理。

初始化包括以下操作：

1）在 Awake 中调用库函数初始化摄像头设备。

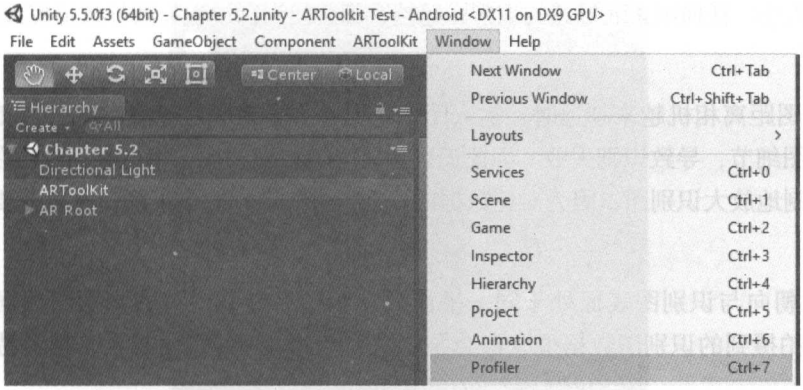
2）在 Enable 中注册 Log 的回调（使库代码的 Log 可以打印到 Unity 的控制台上）。

3）在 Start 中让 ARMarker 载入配置文件或模式文件，并以配置文件初始化摄像机。

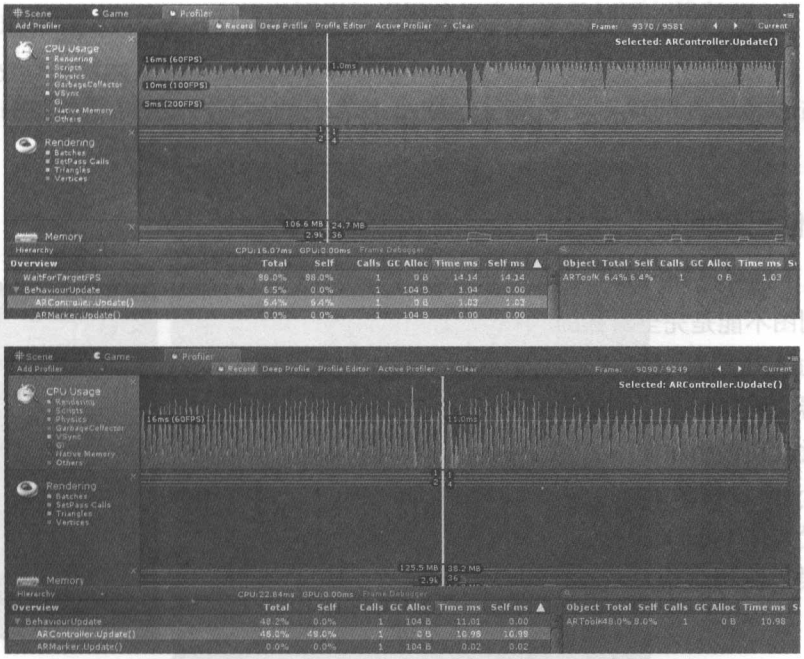
4）在 Update 中动态创建摄像机输入背景，识别图像并更新 ARCamera 的位置。主要处理逻辑在 UpdateAR 中。

5.4.2 ARToolKit 中性能问题的调查

使用 Unity 的 Profiler 可以准确定位性能瓶颈。在 Unity 菜单栏点击 Window → Profiler 打开 Profiler 窗口。



对于ARToolKit来说，摄像机的拍摄尺寸极大地影响着性能，下面是640×480像素（上图）与1920×1080像素（下图）的性能对比。



可见像素总数会对性能产生巨大影响。因为图像分析是由脚本完成的，所以由CPU执行。执行时间会随着输入像素的增加而线性增加，这部分取舍是在开发中需要权衡的：是否应该更多地应用2D-Barcode取代特征点识别图。

5.4.3　ARToolKit的使用限制

基于视觉的追踪可以应用到很多场景，但是这种追踪方式本身存在一定的使用限制。

遮挡

基于视觉的追踪需要识别图位于相机的视野中，这不但限制了相机的移动范围，而且限制

了识别图的大小。任何因素而导致的识别图遮挡，都会使识别过程失败导致追踪虚拟对象消失。

范围

当识别图距离相机越来越远时，输入图像占据的像素也越来越少，这会使相机无法获取足够的识别图细节，导致识别失败。因此识别图越大，识别的成功率越高。然而在实际使用中也不能无限制地放大识别图，因为对比数据的增加也会导致性能的下降，所以需要权衡利弊。

视角

当相机朝向与识别图表面处于同一平面时，明显失去了所有的识别图数据。当视角过小，摄像机拍摄到的识别图数据很少也会导致识别失败。因此视角越大，越容易识别。

光照条件

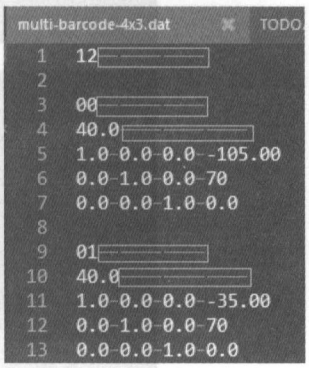
强度过高的光照会导致反光或过曝，导致输入的识别图颜色不准确，进而影响识别的准确率。同理，过于昏暗的光照会丧失很多细节干扰识别处理。建议尽量在平行光的条件（例如太阳光）下追踪效果，同时需要注意避免阴影对识别的干扰。

5.5 跨平台开发的注意事项

5.5.1 插件已知问题

存在以下插件已知问题：

- ❑ 识别图不能是完全对称的
- ❑ 改动 ARMarker 属性时，场景无法保存（应该是由于编辑器代码执行了一些操作但是并未 SetDirty），可以通过少量改动再删除保存
- ❑ Unity 项目目录 Assets\StreamingAssets 的多重识别图的配置文件（.dat）文件中存在不需要的缩进导致读取文件错误，使用时应该删除（插件的 5.3.2 版本中存在此问题）

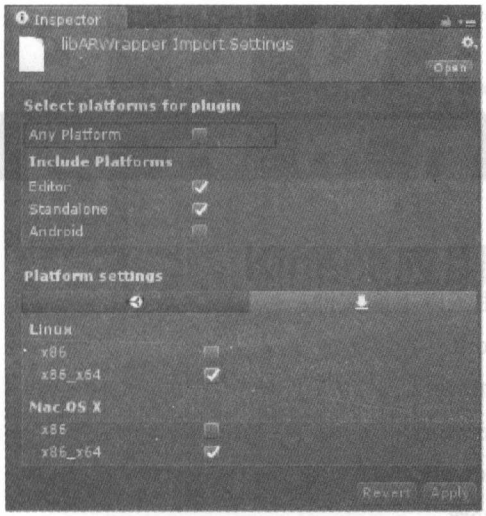


5.5.2 Android

在编译 Android 的 apk 安装包时会出错，具体错误为：

Found plugins with same names and architectures, Assets/Plugins/Android/libARWrapper.so (ARMv7) and Assets/Plugins/x86/libARWrapper.so (AnyCPU). Assign different architectures or delete the duplicate.

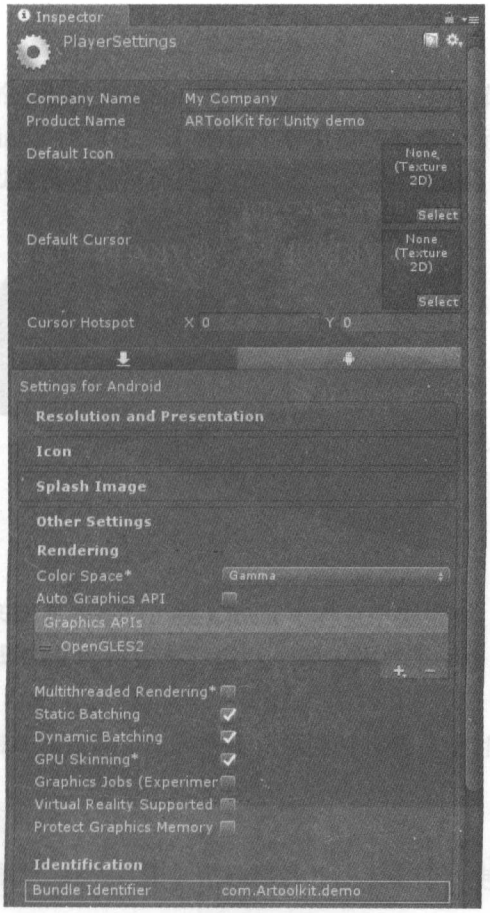
原因是各平台的库文件发生了冲突，在 Assets/Plugins/x86/libARWrapper.so 和 Assets/Plugins/x86_64/libARWrapper.so 库文件选择的全平台，与 Android 平台的库文件冲突。所以取消勾选这两个库文件的 Any Platform，并选择对应的平台类型（x86 或 x86_64）。



使 AndroidManifest 的 package 和应用的 package 保持一致。

在 PlayerSettings 中需要设置 Bundle Identifier。

在 Unity 菜单栏中依次选择 Edit → Project Settings → Player 打开 PlayerSettings。



打开 Assets\Plugins\Android\AndroidManifest.xml，使 package 和上述 Bundle Identifier 保持一致。

```
AndroidManifest.xml   TODO.txt
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.Company.ProductName">
```

5.5.3 iOS

确保在 XCode 中引用了以下库文件：

- Accelerate.framework
- AudioToolbox.framework
- AVFoundation.framework
- CoreGraphics.framework
- CoreMedia.framework
- CoreVideo.framework
- OpenGLES.framework
- QuartzCore.framework
- libjpeg
- libstdc++.6

第 6 章 Chapter 6

Kinect 应用开发

6.1 Kinect 简介

6.1.1 Kinect 是什么

Kinect 是由微软开发的，起初是 Xbox 360 和 Xbox One 的周边体感控制器（目前已经可以在 PC 上使用）。利用 Kinect 使得用户不需要使用手持控制器，而是使用语音指令或手势与应用内容交互，同时可以捕捉用户身体的动作。

Kinect 是基于摄像头来捕捉动态的，在 Kinect 中有三个摄像头：一个 RGB 彩色摄像头，以及红外线发射器和红外线 CMOS 摄像头，后两个镜头构成了一个 3D 结构光深度感应器，由此识别深度信息。

Kinect 中也内建了阵列式麦克风，由多组麦克风同时收音，比对后可以消除杂音。

6.1.2 Kinect 功能特性简介

Kinect 的动作捕捉功能极其强大，附加微软提供的技术支持，使用户可以十分便利地制

作体感应用。Kinect 的主要功能如下：

- ❑ 人体姿势识别与追踪
- ❑ 深度检测
- ❑ 面部表情识别与追踪
- ❑ 手势识别
- ❑ 语音识别

通过以上这些主要功能，开发者可以获取更多、更精准的用户输入，从而增强用户的 AR 体验。

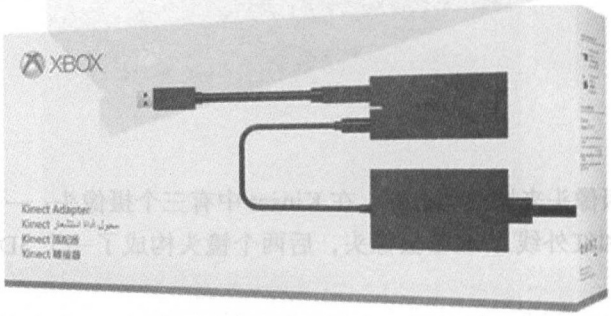
6.2 搭建 Kinect 的 Unity 3D 开发环境

6.2.1 硬件需求

搭建 Kinect 的 Unity 3D 开发环境的硬件需求包括：

- ❑ 安装有 Windows 8 或更新版本的 PC
- ❑ Kinect V2 一台

- ❑ Kinect PC 适配器



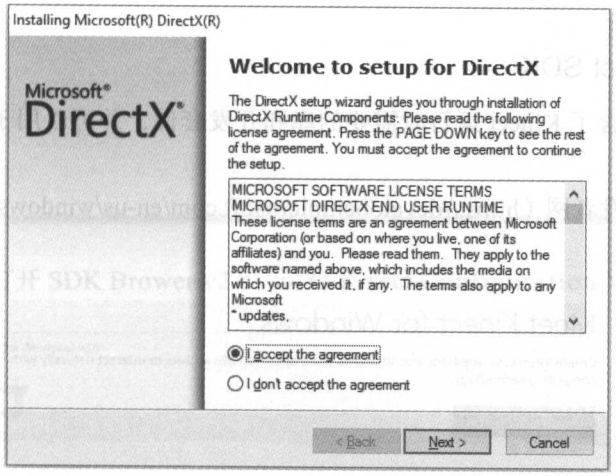
- ❑ 双核 3.1 GHz 64 位处理器或更高级的处理器
- ❑ 4GB 或以上的内存
- ❑ 可以适配 Kinect 的 USB 3.0 接口
- ❑ 支持 DirectX 11 的显卡

满足硬件需求后，再安装 Kinect 开发环境。

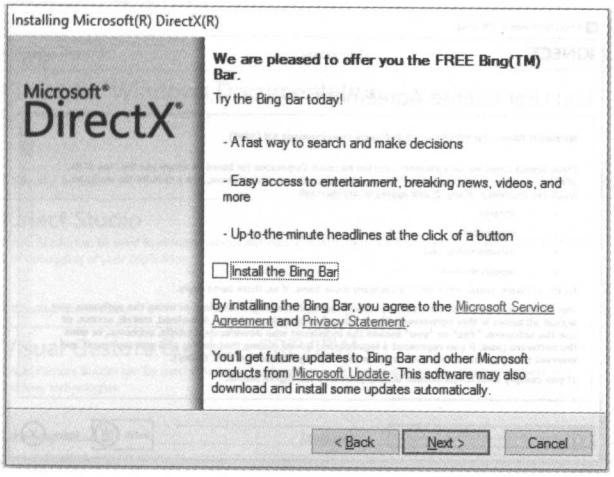
6.2.2 安装 DirectX

由于 Kinect 在 PC 上需要调用显卡加速，因此需要安装 DirectX 11 的运行时环境。建议使用 Web Installer 安装确保获取最新版本。微软官方下载地址为 https://www.microsoft.com/en-us/download/details.aspx?id=35。

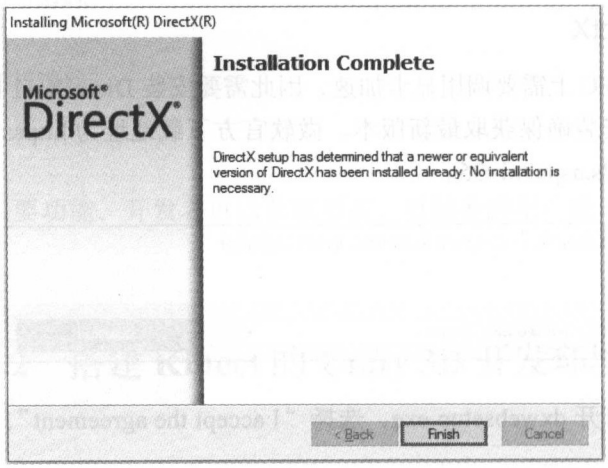
下载完成后，打开 dxwebsetup.exe，选择“I accept the agreement”，点击 Next。



选择是否安装必应工具条，点击 Next。



等待安装完成后，点击“Finish”。

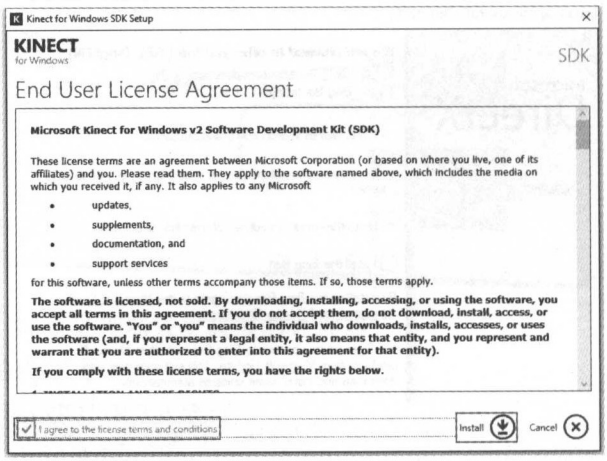


6.2.3 安装 Kinect SDK

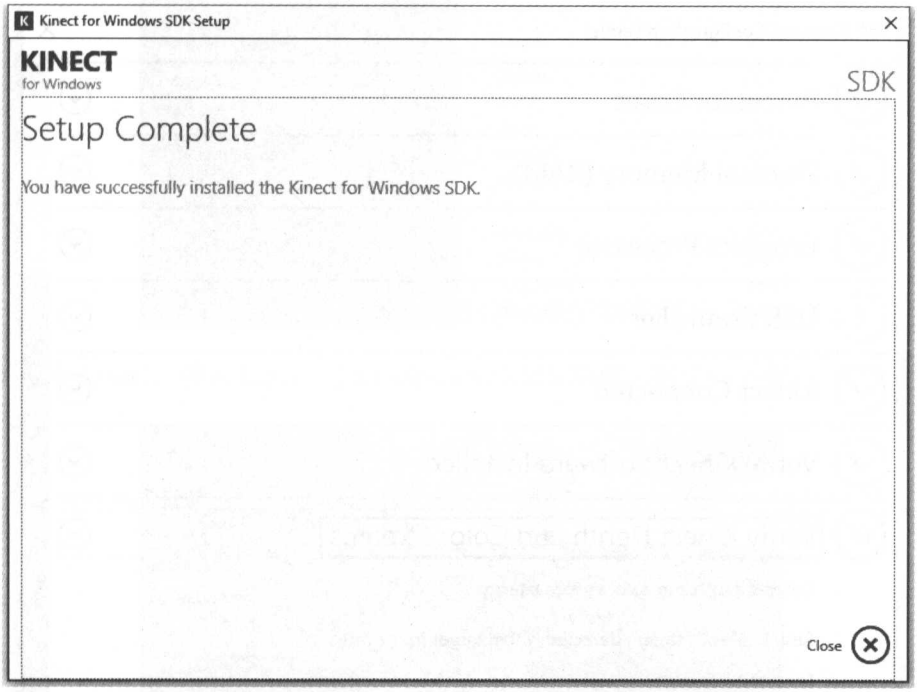
Kinect SDK 包含了 Kinect 运行时需要的库和开发套件，并可以用于检测 Kinect 是否正常工作。

登录 Kinect 开发者网（https://developer.microsoft.com/en-us/windows/kinect）站可以找到最新的 SDK 安装包。

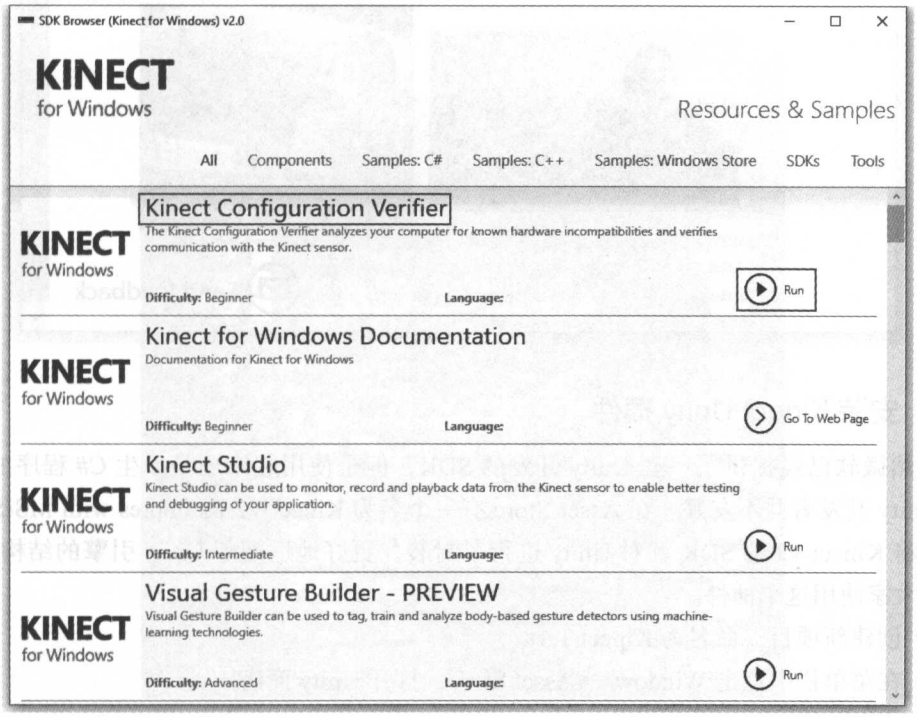
目前最新的版本为 KinectSDK-v2.0_1409-Setup.exe，双击运行，选中“I agree to the license terms and conditions”，然后点击 Install。



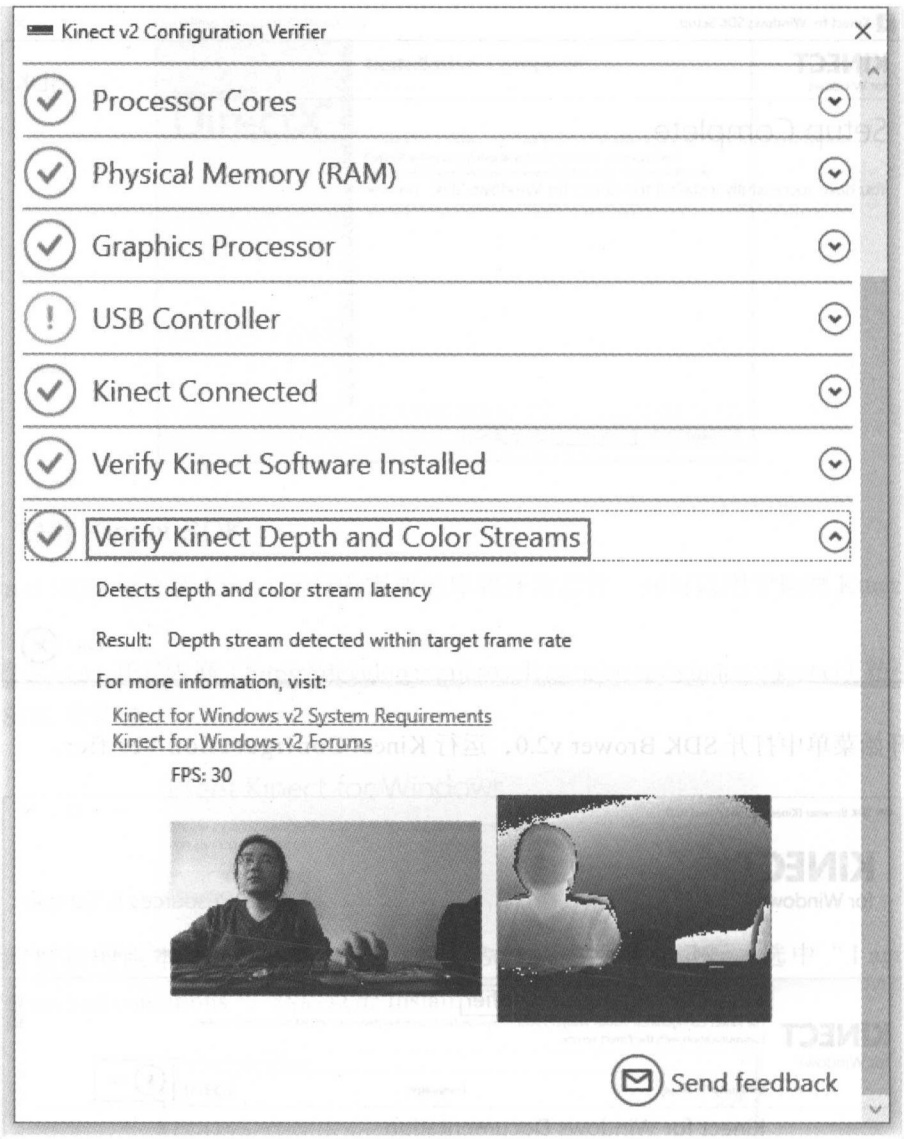
等待安装完成后，点击 Close 关闭安装向导。



在开始菜单中打开 SDK Brower v2.0，运行 Kinect Configuration Verifier。



打开 Verify Kinect Depth and Color Streams，如果看到图像代表驱动成功。

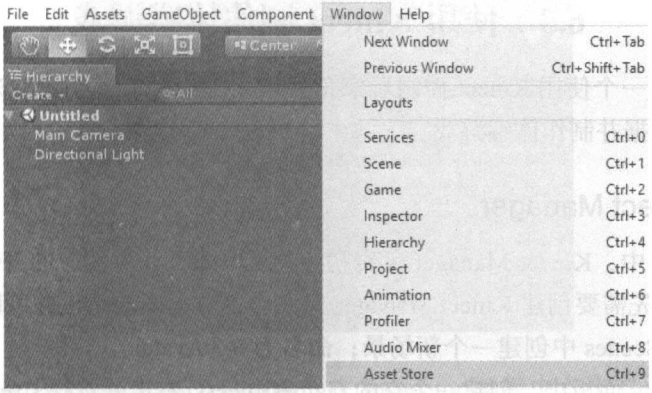


6.2.4 安装 Kinect Unity 插件

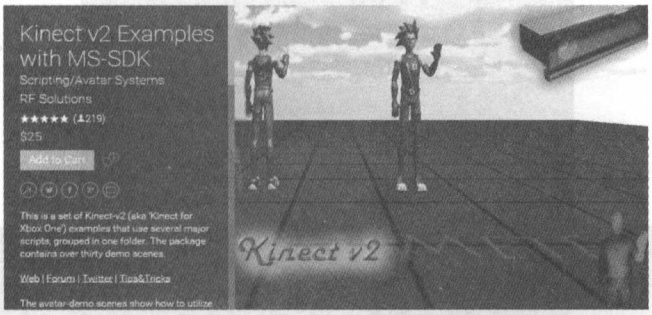
虽然微软已经提供了一套 Unity 开发的 SDK，但是使用起来还是原生 C# 程序的方式，对于 Unity 开发者并不友好。在 Asset Store 有一个名为 Kinect v2 Examples with MS-SDK 的插件，将 Kinect v2 的 SDK 针对 Unity 进行了封装，更好地适配了 Unity 引擎的结构。在这里推荐大家使用这个插件。

1）创建新项目，命名为 Kinect Test。

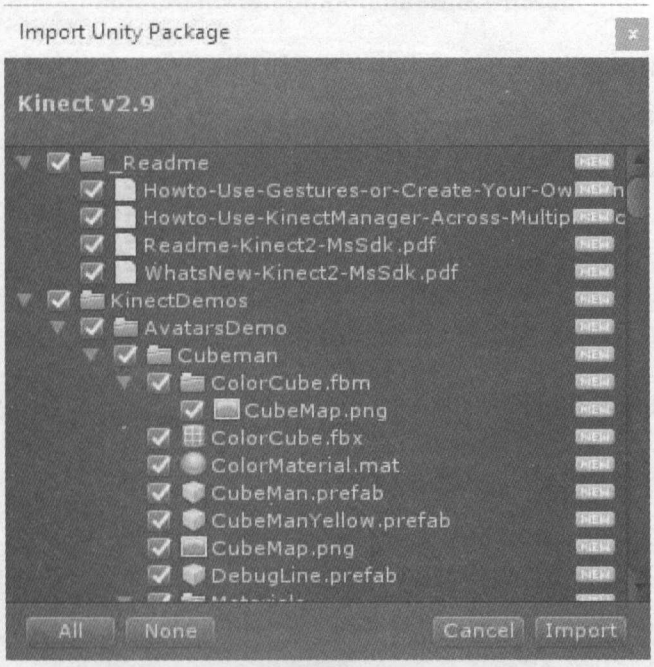
2）在菜单栏中点击 Window → Asset Store，打开 Unity 商店。



3）搜索 Kinect，进入 Kinect v2 Examples with MS-SDK，购买后下载。



4）下载完成后，点击 Import 导入 Unity。



6.3 使用 Kinect 制作体感游戏

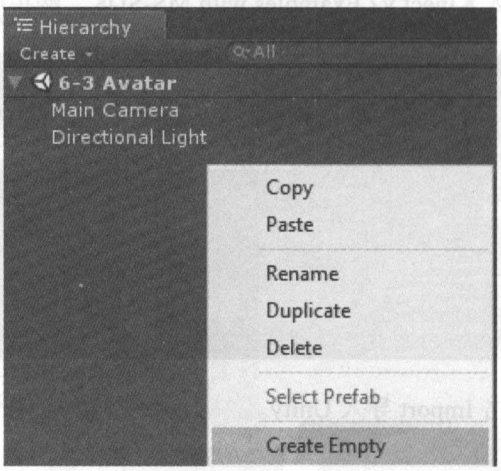
本节通过制作一个使用 Kinect 控制玩家角色消灭附近敌人的小游戏来了解一下如何获取 Kinect 的输入数据并制作体感游戏。

6.3.1 创建 Kinect Manager

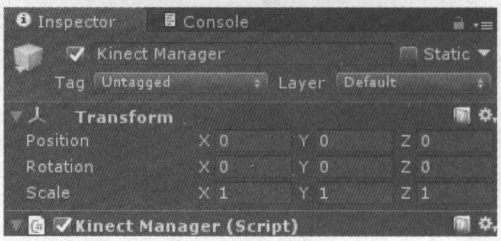
在 Kinect 插件中，Kinect Manager 负责初始化 Kinect，获取用户的身体输入数据。在创建任何对象前，首先需要创建 Kinect Manager 保证设备被正确初始化并可以正常工作。

1）在 Assets/Scenes 中创建一个新场景；命名为 6-3 Avatar。

2）在 Hierarchy 面板中，创建一个空的 GameObject：右击并选择 Create Empty。



3）重命名为 Kinect Manager，选中并为其添加 Kinect Manager 组件。



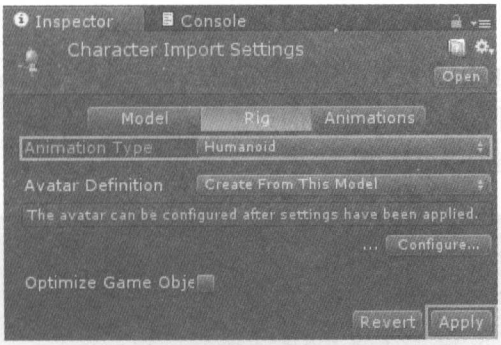
保留 Kinect Manager 中数据的默认值。

6.3.2 导入人物 3D 模型并创建 Avatar

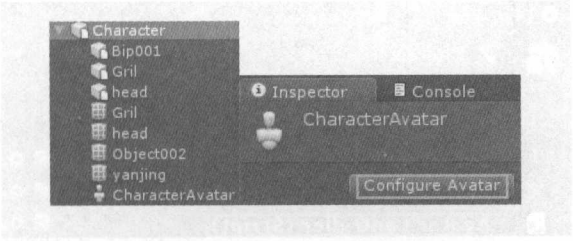
1）在 Assets 文件夹下创建 Models 文件夹。

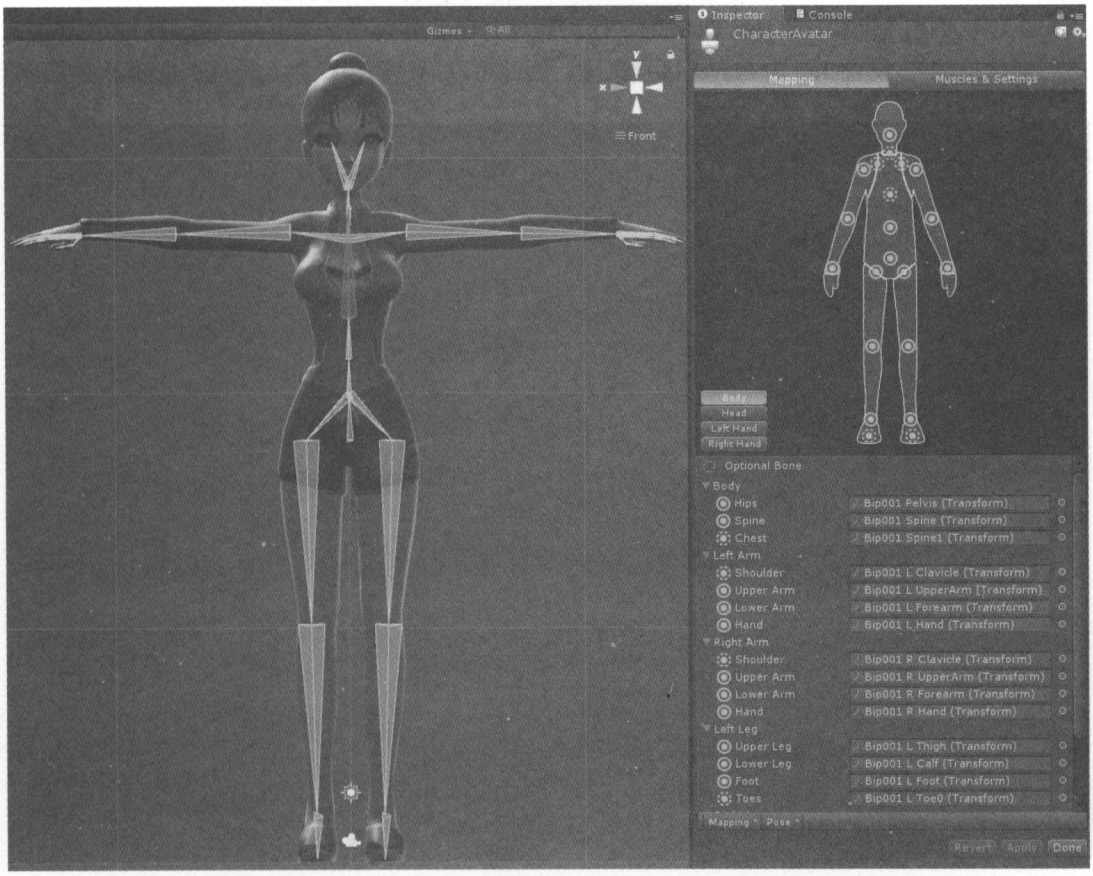
2）将 Character 模型文件拖入 Models 文件夹。

3）选中 Character，在 Inspector 面板中将 Animation Type 设置为 Humanoid，点击 Apply。



4）点击 Character 下的 CharacterAvatar，在 Inspector 面板中点击 Configure Avatar，确认每个骨骼的位置和模型的骨骼位置一一对应。

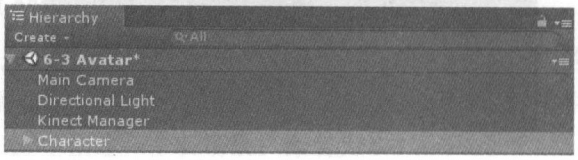




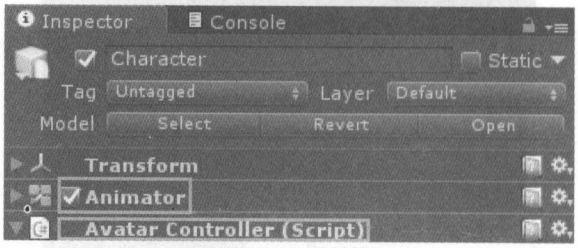
6.3.3 创建人物

Avatar Controller 将 Kinect 获取的用户骨骼和人物本身的 Avatar 骨骼绑定在了一起，因此用户输入可以直接控制与其绑定的模型的动作。

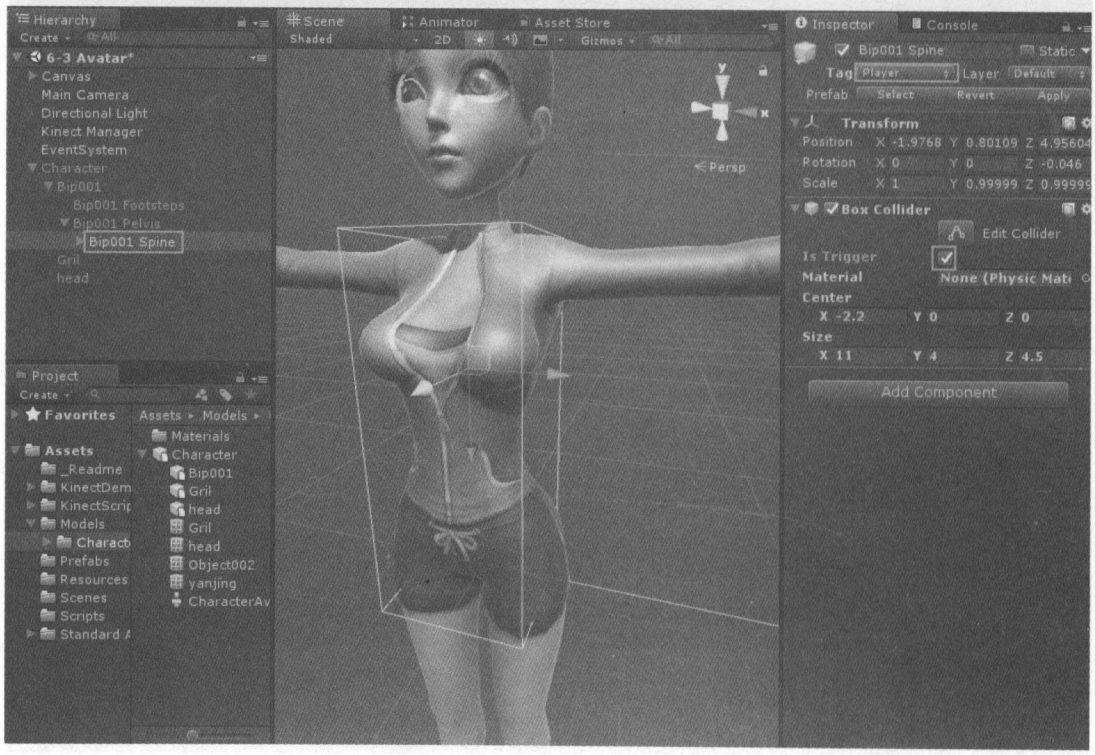
1）将模型拖至 Hierarchy 面板中。



2）为其添加 Avatar Controller 组件，需要确保组件和 Animator 在同一级。



3）为人物的身体添加 Box Collider，检测碰撞并调整其大小，选中 Is Trigger（通过 Trigger 进行碰撞检测），设置 Tag 为 Player 区别碰撞，因此可以通过碰撞检测人物是否遭到了攻击。



4）点击运行，站在 Kinect 前，让 Kinect 可以捕捉到你的全身，可以发现人物跟着你运动，Avatar Controller 的作用就是获取与其同级的 Animator 组件后，将 Kinect 捕获的输入数据与人物的 Avatar 骨骼绑定后同步两者的运动。最终使人物随着玩家一起运动。

5）在 Assets 目录下创建 Scripts 文件夹，创建 Player.cs，添加以下代码：

```
using UnityEngine;

public class Player : MonoBehaviour
{
    public int health = 100;
    public Transform hitTrans;
    public static Player instance;

    void Awake()
    {
        instance = this;
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag != "Player") {
            health -= 10;
            if (health <= 0) {
                Destroy(gameObject);
            }
    }
}
}
```

首先定义玩家角色的血量，在 OnCollisionEnter 方法中，当检测到碰撞时，血量都减少 10。当血量为 0 时，销毁玩家对象。

然后定义玩家的攻击目标的 Transform，用于敌人识别玩家的位置。

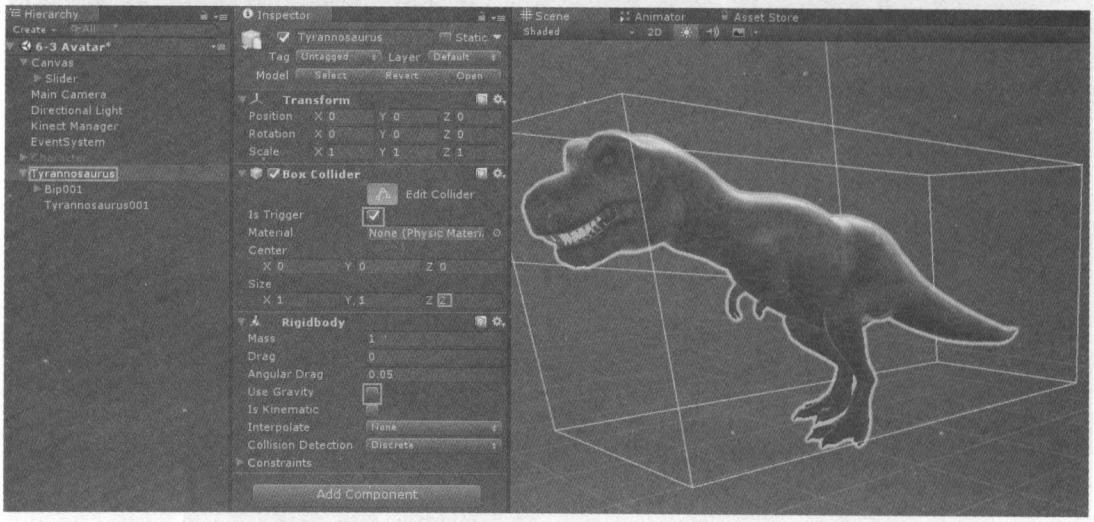
之后定义玩家角色的实例，由于玩家是唯一的，所以这里定义为 static。在 Awake 中将本实例初始化。由此可以方便地获取玩家角色的引用。

6）将 Player 拖至人物身上，并添加 Hit Trans 的引用。

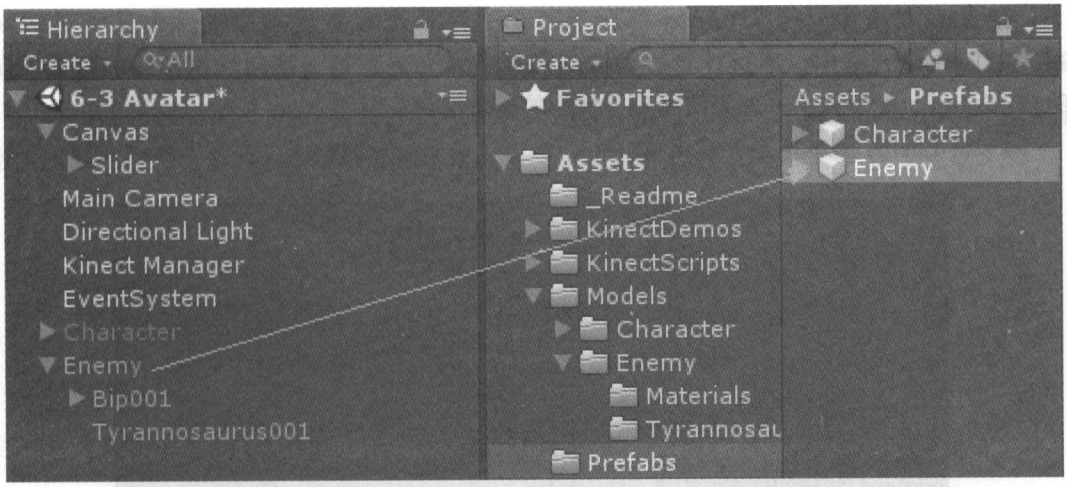
7）为 Player 添加 Rigidbody 组件，并取消勾选 Use Gravity。

6.3.4 创建敌人

1）将 Enemy 文件夹拖至 Assets/Models 文件夹下，将 Tyrannosaurus 拖至 Hierarchy 面板。删除 Animator 组件，并为其添加 Box Collider 和 Rigidbody 组件，取消勾选 Use Gravity 属性。



2）将 Tyrannosaurus 改名为 Enemy，拖至 Assets/Prefabs 目录创建 Prefab，并删除 Hierarchy 中的 Enemy。

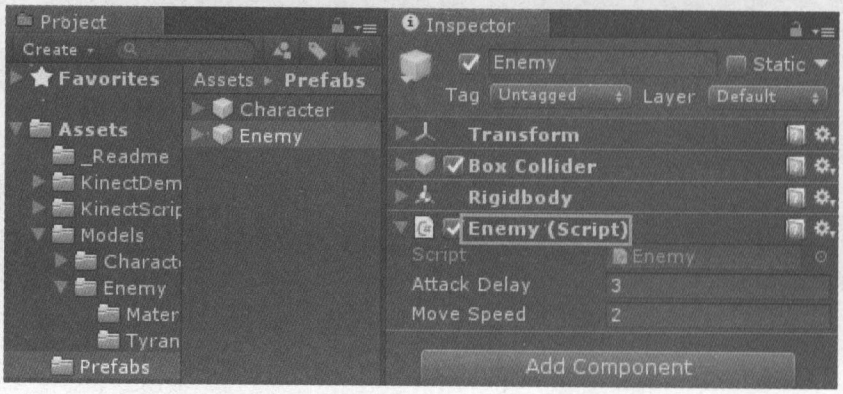


3）在 Assets/Scripts 目录下创建 Enemy.cs，添加以下代码：

```
using UnityEngine;
using System.Collections;

public class Enemy : MonoBehaviour
{
    public float attackDelay = 3;
    public float moveSpeed   = 2;

    bool isMoving;

    IEnumerator Start()
    {
        yield return new WaitForSeconds(attackDelay);

        isMoving = true;
    }

    void Update()
    {
        if (isMoving) {
            Vector3 playerPosition = Player.instance.hitTrans.position;
            Vector3 moveDirection = playerPosition - transform.position;
            float moveAmount = moveSpeed * Time.deltaTime;
            Vector3 moveVector = moveAmount * moveDirection.normalized;
            transform.Translate(moveVector);
        }
}

    void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
}
}
```

在 Start 方法中，每个 Enemy 都会等待 3 秒后才开始向玩家移动，也就是进攻玩家。在 Update 方法中，通过计算移动方向和速度，来更新每一帧向玩家移动的距离。在 OnCollisionEnter 事件中，如果检测到与玩家的碰撞则自动销毁。

4）将 Enemy 脚本拖至 Prefab 上。



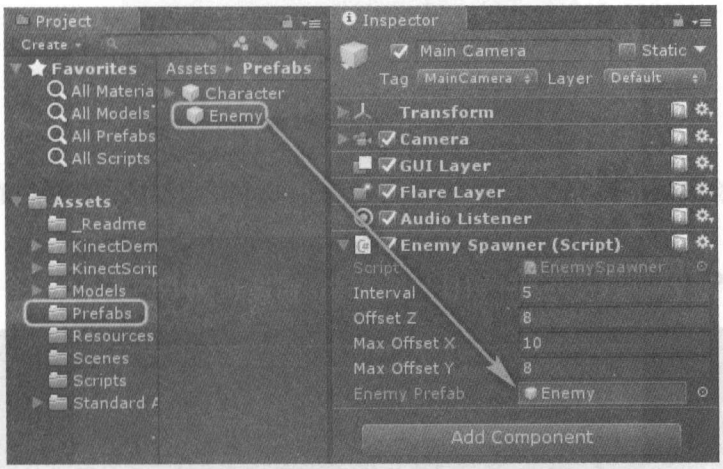
5）在 Assets/Scripts 目录下创建 EnemySpawner.cs，添加以下代码：

```
using UnityEngine;

public class EnemySpawner : MonoBehaviour
{
    public float interval   = 5f;
    public float offsetZ    = -8f;
    public float maxOffsetX = 10f;
    public float maxOffsetY = 8f;
    public GameObject enemyPrefab;

    float passedTime;

    void Update()
    {
        passedTime += Time.deltaTime;
        if (passedTime >= interval) {
            passedTime = 0;
            SpawnEnemy();
        }
    }

    void SpawnEnemy()
    {
        GameObject spawnedEnemy = Instantiate(enemyPrefab);

        Vector3 enemyPosition = Player.instance.hitTrans.position;
        enemyPosition.x += Random.Range(-maxOffsetX, maxOffsetX);
        enemyPosition.y += Random.Range(-maxOffsetY, maxOffsetY);
        enemyPosition.z += offsetZ;
        spawnedEnemy.transform.position = enemyPosition;
```

```
    }
}
```

在 Update 方法中，每 5 秒会生成一个敌人，在生成敌人时，使用 Random 类将生成的敌人放置在角色面前的一个随机位置。

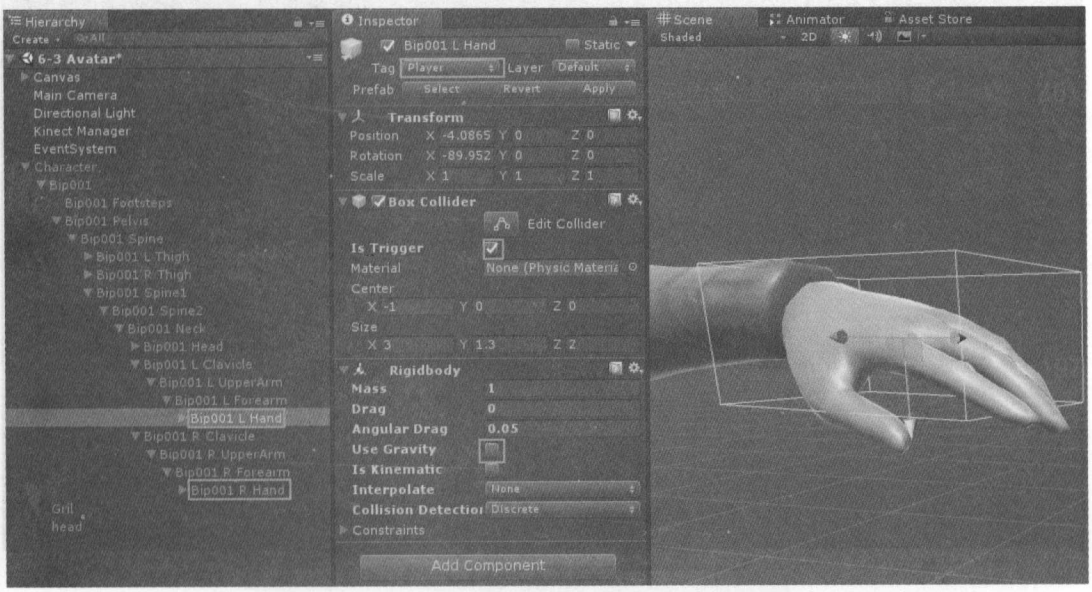
6）将 Enemy Spawner 拖至 Main Camera，并将 Enemy 的 Prefab 拖至对应位置。



7）点击运行按钮，运行一段时间可以发现角色在经过 10 次碰撞后将会死亡。

6.3.5 为人物添加攻击处理

1）为人物的左手和右手添加 Box Collider 和 Rigidbody 检测碰撞，并将其的 Tag 置为 Player。因此可以通过碰撞检测判断是否击中了敌人。

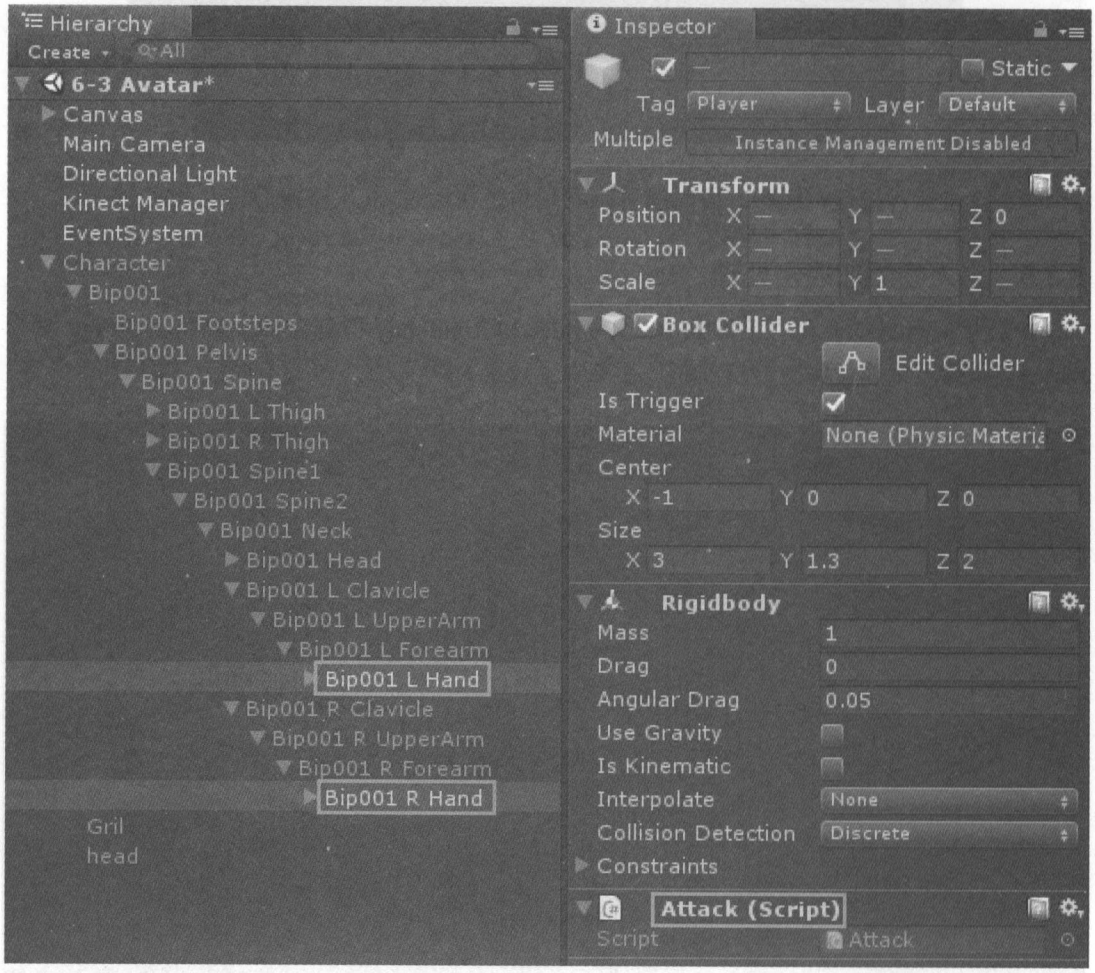


2）在 Assets/Scripts 目录下添加 Attack.cs 文件，并添加以下代码：

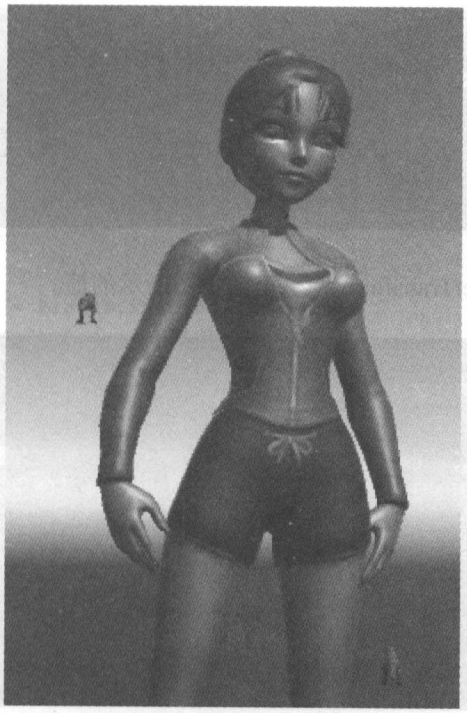
```
using UnityEngine;

public class Attack : MonoBehaviour
{
    void OnTriggerEnter(Collider other)
    {
        if (other.tag != "Player") {
            Destroy(other.gameObject);
        }
    }
}
```

这里只要检测到不是 Player 的部分，就直接销毁该对象，这样就可以直接销毁 Enemy 对象。

3）将 Attack 拖至添加了 Collider 的左手和右手上。



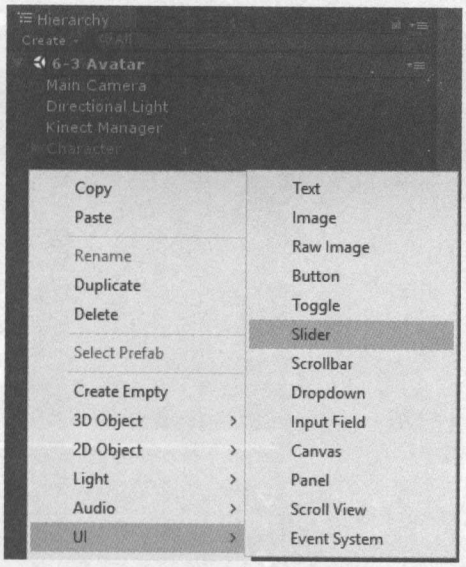
运行后，使用左手和右手攻击敌人。



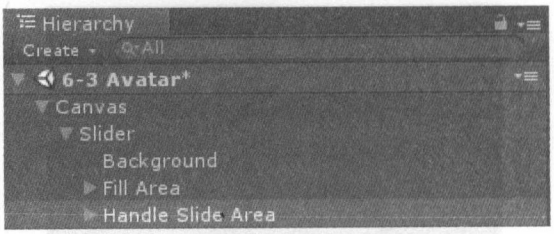
6.3.6 添加 UI 显示

我们制作一个血条来显示角色的血量。

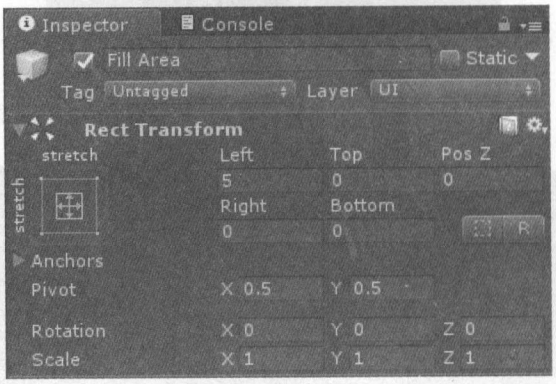
1）在 Hierarchy 上右击，并选择 UI → Slider。



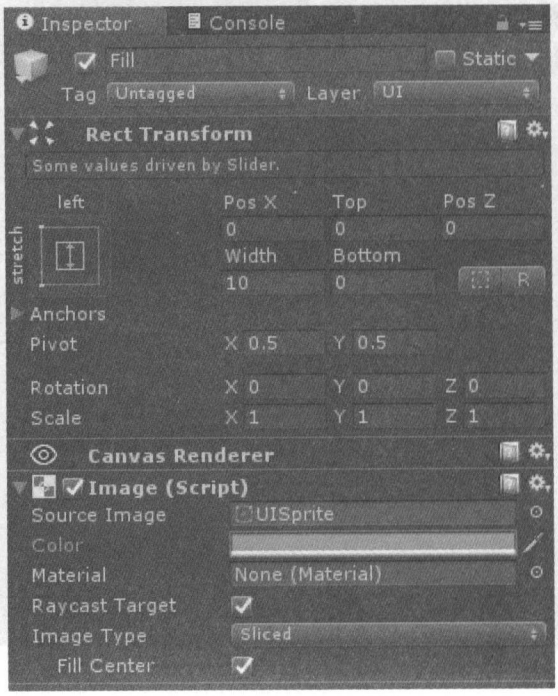
2）删除 Slider 下的 Handle Slide Area。



3）将 Fill Area 的 RectTransform 调整至如下图所示的值。



4）将 Fill 的颜色改为红色，调整血条的位置。

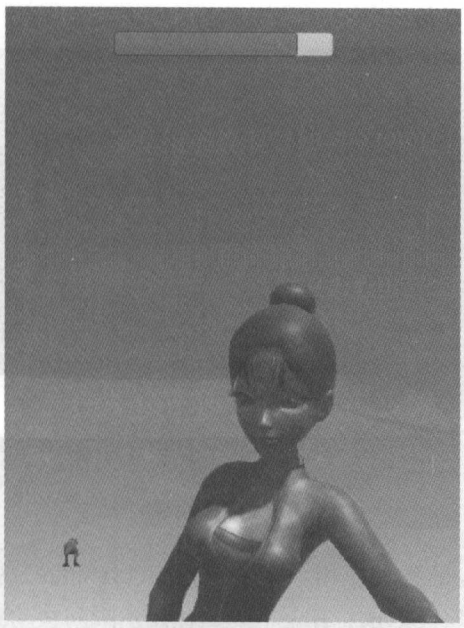


5）在 Assets/Scripts 目录下添加 HealthBar.cs，添加以下代码：

```
using UnityEngine;
using UnityEngine.UI;

public class HealthBar : MonoBehaviour
{
    float initHealth;
    Slider slider;

    void Start()
    {
        initHealth = Player.instance.health;
        slider = GetComponent<Slider>();
        if (slider) Debug.LogError("Slider is missing!");
    }

    void Update()
    {
        slider.value = Player.instance.health / initHealth;
    }
}
```

首先在 Start 方法中获取角色血量初始值，然后在 Update 方法中更新角色的血量。

6）运行程序，观察受伤后血条的变化。

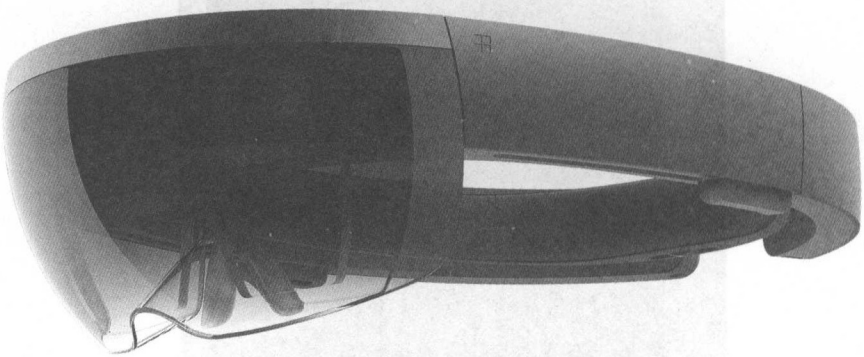


Chapter 7 第7章

HoloLens

7.1 HoloLens 简介

HoloLens 是由微软公司在 2015 年 1 月 21 日与 Windows 10 同时公布的智能眼镜产品。它采用 Windows 10 系统，拥有先进的传感器、高清晰度 3D 光学透镜显示器，以及环绕音效，允许用户在增强现实的场景中通过凝视（Gaze Input）、语音输入（Voice Input）以及手势（Gesture）与虚拟世界进行交流。



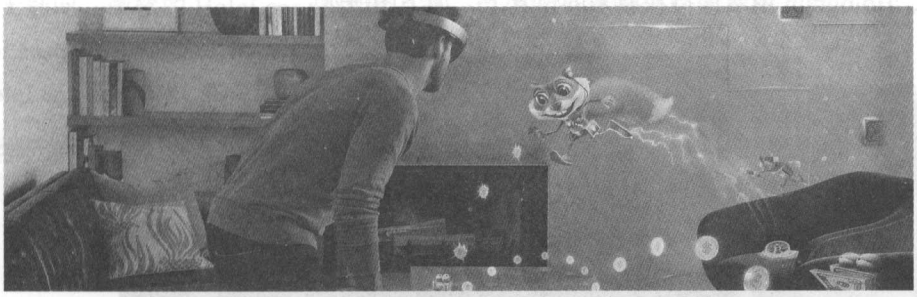
HoloLens 外观

微软展示了 HoloLens 的各种应用。其中包括 HoloStudio（一个 3D 建模软件），可以输出适用于 3D 打印机的模型；Skype；HoloTour，让用户瞬间移动到不同地点，从而获得身临其境的体验，该应用可以让用户看到诸如罗马和马丘比丘这样的高清 360 度全景展示，可以通过在某个地点随意走动，同时接触虚拟物体，从而享受一趟虚拟旅程。游戏 Young Conker 则将虚拟的松鼠带到了现实世界中。

HoloStudio

Skype

HoloTour



Young Conker

接下来我们将详细介绍 HoloLens 的特性及使用方法。

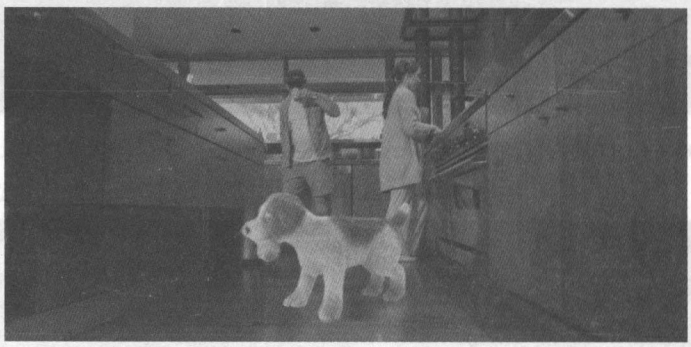
7.1.1 Hologram 简介

HoloLens 中创建的 Hologram（全息图）是一种包含光线和声音的物体，这种物体可以出现在你周围的真实环境中，Hologram 跟真实物体一样能够对用户的手势、语音等动作做出响应，并且能够与真实世界的表面产生互动，我们可以借助 Hologram 在真实世界中创造数字物体。

Hologram 在渲染的过程中，是向真实的世界中增加了光线，也就是说用户通过 HoloLens 的全息透镜，不但可以观察到由 HoloLens 渲染的 Hologram，也可以看到真实的世界。HoloLens 不能将光线从用户的眼睛移除，这就意味着，Hologram 没有办法渲染出黑色的物体，黑色的内容在 HoloLens 中的效果是透明的，同时这也是我们在后面章节中将 Camera 背景颜色设置为黑色的原因所在。

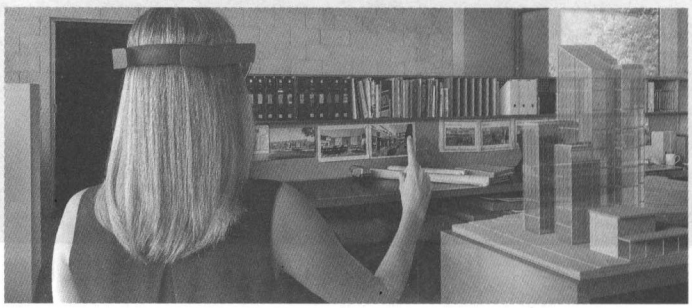
每一个 Hologram 都可以为其添加音效，而这种音效是从 Hologram 在所处空间的真实位置上发出的，HoloLens 的这种空间环绕音效是通过两个耳朵上方的扬声器实现的。

下图中站在地板上的宠物狗就是一个 Hologram，而图中其他信息都是真实的场景。

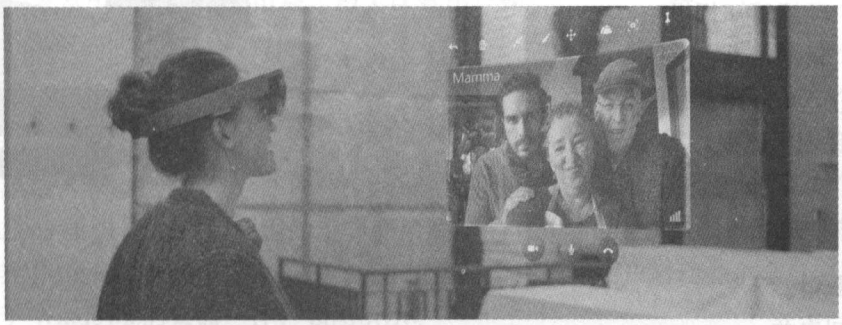


Hologram 在真实世界中有两种处理方式。

第一种是放置，用户可以非常精确地将一个 Hologram 放置在某个位置，不论从哪个角度去观察它，它都会非常稳定地显示在真实场景中，不会发生抖动或丢失的现象。而且，一旦用户为 Hologram 添加了空间锚（Spatial Anchor），就可以将 Hologram 钉在固定的位置，这个时候用户离开房间一段时间之后再回来，会发现系统已经记忆了 Hologram 刚才的位置和操作，Hologram 依然保持在原来的位置上，如下图所示：



第二种处理方式是跟随，我们可以设置某个 Hologram 的位置总是相对于用户的位置，比如一直保持在用户面前一定距离，或者跟随在用户身后，如下图所示的 Skype，用户跟亲友视频通话的过程中，视频窗口会一直保持在用户视线前方。



Hologram 不仅仅是一个将光线和声音融合的数字物体，更为激动人心的是它可以和真实的世界进行互动，因为 HoloLens 知道每一个 Hologram 在真实场景中的位置，所以用户可以通过凝视、手势或语音的方式发出指令，而我们可以在程序中去实现 Hologram 针对每一种指令所产生的回应，就像我们在游戏开发中所做的那样。

除了可以和用户进行互动之外，Hologram 还可以跟它所在的真实场景产生互动，因为 HoloLens 会在程序进入时扫描真实环境的表面，并产生网格信息（也就是 3D 软件中的 Mesh），所以我们可以实现将一个 3D 的虚拟篮球（Hologram 表现为 3D 篮球）放置在空中让其做自由落体，当篮球与真实场景地面的网格发生碰撞时会弹回空中。而在用户看来，是虚拟的篮球与真实的地面发生了碰撞。对于非程序开发者或专业人士，这无疑是一种神奇的体验。

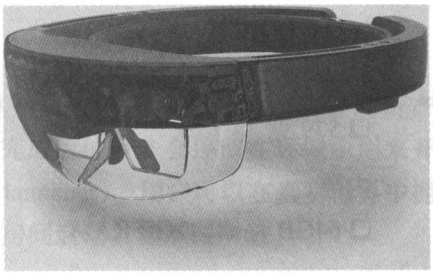
既然可以实现场景的扫描与建模，我们自然可以将一个虚拟物体放置在真实的物体后面，这会产生真实物体遮挡了虚拟物体的效果，这种遮挡效果是很难在其他智能眼镜中看到的强大功能。

作为 HoloLens 的开发者，可以尽情发挥想象力，创造出更多的 Hologram，本章将会逐步介绍 HoloLens 开发过程的具体步骤和原理。

7.1.2 HoloLens 硬件细节

毫无疑问，微软的 HoloLens 是世界上第一台功能完整的全息计算机，本节我们简要列举 HoloLens 的一些硬件细节。

HoloLens 设备中包含以下主要模块。



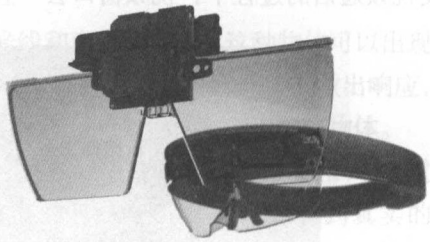
光学元件

- ❑ 全息透视镜（波导）
- ❑ 两个高清光线引擎
- ❑ 自动瞳孔距离校准
- ❑ 全息分辨率

❑ 全息密度

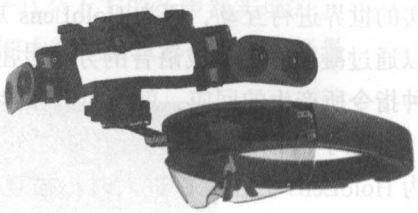
传感器

❑ 1 个惯性测量单元（IMU）
❑ 4 个环境感知相机
❑ 1 个景深相机
❑ 1 个 200 万像素摄像头
❑ 混合现实捕捉
❑ 4 个麦克风
❑ 1 个环境光传感器



人类感知

❑ 空间声音
❑ 凝视追踪
❑ 手势输入
❑ 声音支持



输入 / 输出 / 连接器

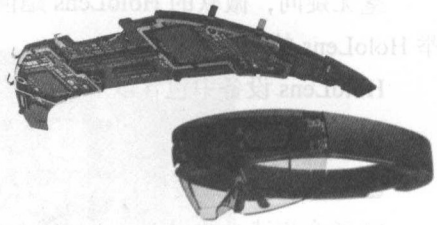
❑ 内置扬声器
❑ 3.5 毫米音频接口
❑ 声音调节
❑ 亮度调节
❑ 电源键
❑ 电源状态指示灯
❑ Wi-Fi 802.11ac
❑ 无线适配器 Micro USB 2.0 接口
❑ 低功耗蓝牙 4.0

电源

❑ 电池续航（2~3 小时正常使用，长达两周待机时间，充电时全功能使用）
❑ 被动冷却（无风扇）

处理器

❑ 英特尔 32 位架构定制的微软全息处理单元（HPU 1.0）



重量

❑ 579 克

内存

❑ 64GB 存储，2GB RAM

系统

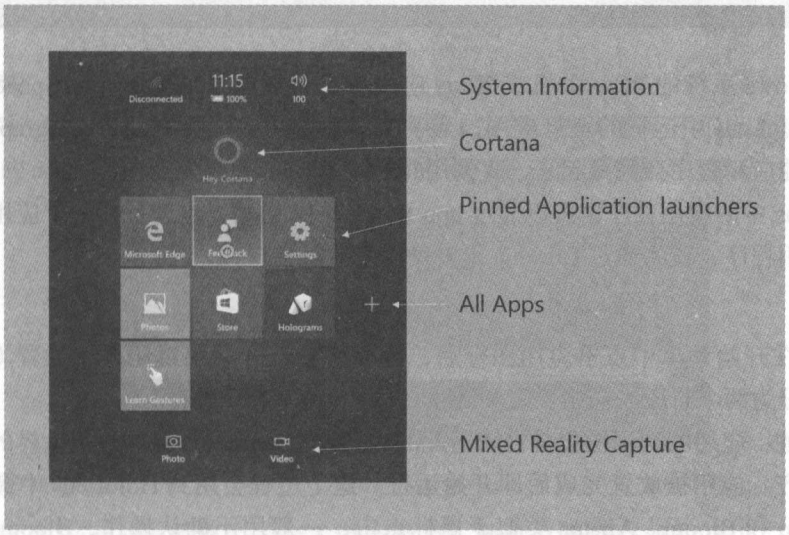
❑ Windows 10

另外，我们在购买了 HoloLens 开发版之后，在 HoloLens 包装盒中会有一个点击器，用于改善用户操作，如右图所示。

7.1.3 HoloLens shell

开始菜单

HoloLens shell 是由我们周围的世界和系统级的 Holograms 组成，我们将这个空间称为混合的世界 (Mixed World)，HoloLens Shell 包含一个用来让用户启动各个全息应用的开始菜单，类似于 Windows 的桌面一样，如图所示。



上图中各个部分的功能如下：

❑ System Information: 系统信息，包括 Wi-Fi 状态，电池状态，当前时间和音量。

❑ Cortana：唤起 Windows 10 内置机器人助理按钮。

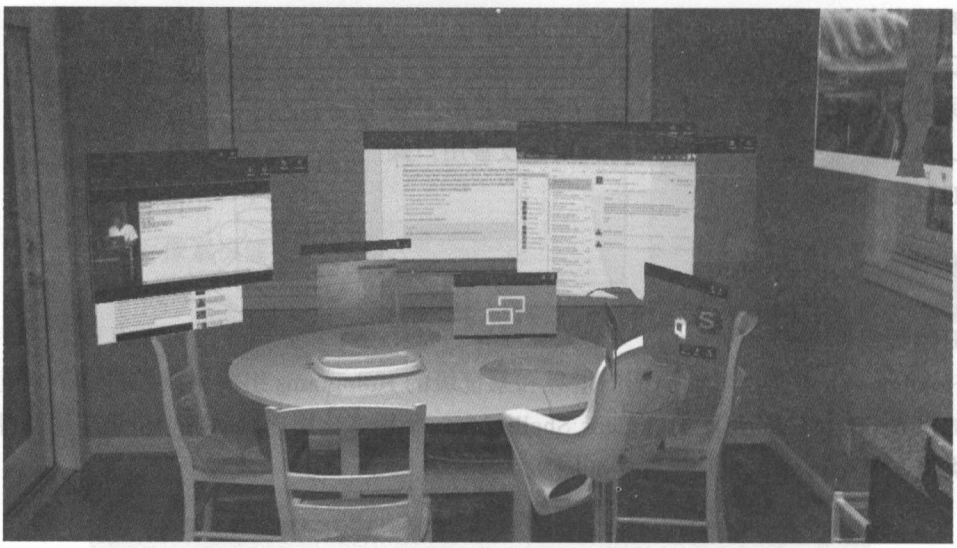
❑ Pinned Application Launchers：应用程序启动界面。

❑ All Apps：打开所有 App 的列表。

❑ Mixed Reality Capture：混合现实拍摄。

运行应用

全息应用程序可以放置在真实世界中，开始菜单是所有应用的总目录，放置的资产（asset）可以是 2D 的面板，也可以是 3D 的模型。放置完成后的资产会停留在环境中用来开启应用，我们可以将这种资产称之为应用启动器（App launcher）。用户可以在真实世界中拥有多个启动器的副本，同一个应用也可以同时在不同的房间启动，如下图所示。



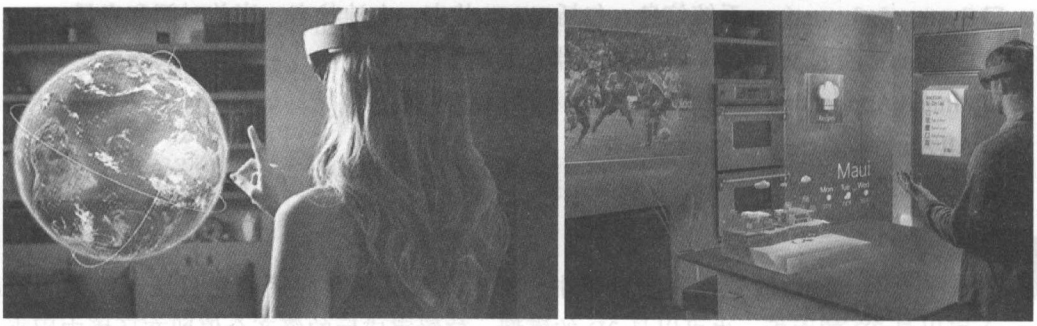
在 Windows 系统的 PC、手机或 Xbox 应用中，可以通过 HolographicSpace API 将其转换为可在 HoloLens 中运行的全息应用，当一个全息应用在全息视图（holographic view）中运行时，其他应用程序会隐藏起来，直到用户通过 Bloom 手势返回主界面。

HoloLens 中的应用程序也可以通过 app-to-app API 启动，也就是在一个应用程序中启动另外一个应用程序，或者通过 Cortana 语音助手唤起其他应用程序。

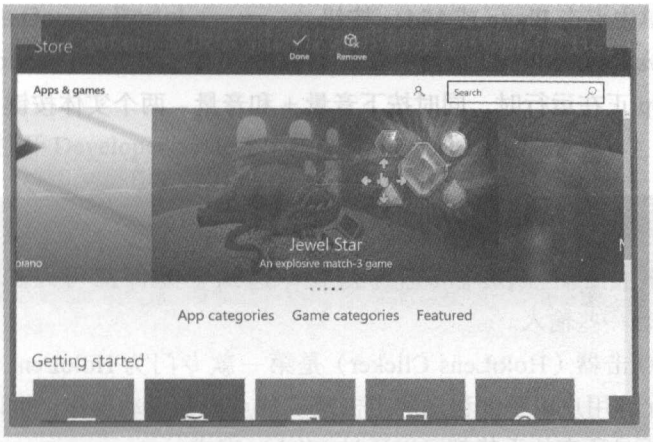
放置

当用户在开始菜单中点击应用图标后，界面将会从开始菜单切换到放置（placement）模式。放置分为两个阶段：初始位置和调整。

初始位置：应用的初识位置是自动适配的，系统会根据用户看向真实世界的方向在尺寸和位置上对齐。应用被放置完成后即开始运行。这个过程会用到 HoloLens 中最重要的两种手势：Air-tap 和 Bloom。Air-tap 类似于鼠标点击，一般用于确认操作，Bloom 则一般用来取消或回到主界面。两种手势的操作示意如下图所示：



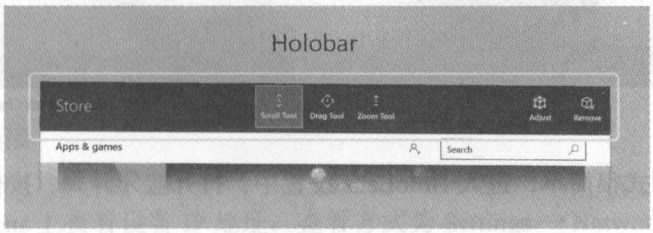
调整：每一个应用都会有一个应用程序栏（App bar），点击应用程序栏的“Adjust”按钮，将会看到以下效果：



在这种模式下，我们可以用手势拖动改变视图的大小或者移动其位置。3D 的 Holograms 还可以被旋转。调整合适之后可以点击“Done”按钮或者发送“Done”语音指令。

应用程序栏

应用程序栏（App bar）是出现在 app 的 2D 视图上方的，默认情况下可以让您进行调整或者移除，不同的 app 可能会有额外属性，比如下图的 app bar 还包含了滚动，拖动，缩放等属性。



Cortana

Cortana 就是 Windows 系统中的个人语音助手，她可以帮您完成在 HoloLens 中的诸多任务，比如启动应用，重启设备，检索信息等等。合理利用 Cortana 可以让您的操作更加轻松便捷。

7.1.4 使用 MRC

混合现实拍摄（Mixed Reality Capture），简称 MRC。HoloLens 能够提供给用户一种混合现实体验，MRC 能够让用户将体验场景以图片 (.jpg) 或视频 (.mp4) 的形式保存下来。拍摄下来的视频或图片不仅仅可以让我们在朋友圈分享体验，还可以指导其他用户某个 App 的操作方式。

获取 MRC 的步骤如下：

1）通过给 Cortana 发送语音指令：“Hey Cortana, take a picture”“Hey Cortana, start recording”分别是告诉 Cortana 截取一张图片和开始录制视频。“Hey Cortana, stop recording”则是停止录制的指令。

2）在开始菜单中点击 Photo 或 Video 按钮。

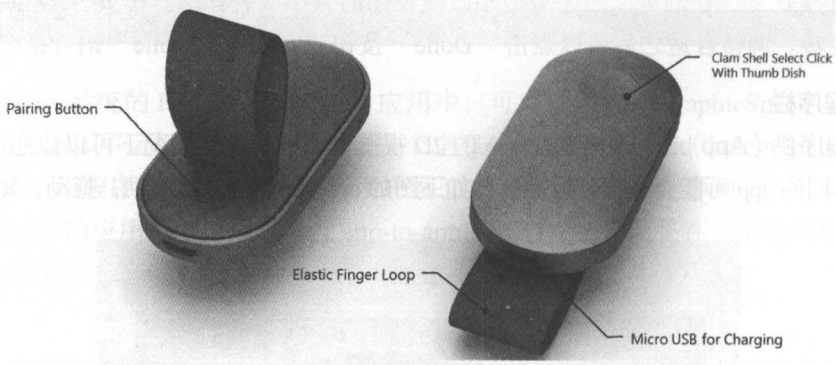
3）在 Windows Device Portal 页面中获取 MRC。

4）当某个 App 正在运行时，同时按下音量 + 和音量 – 两个实体按键，可以截取一张当前视图的图片。

7.1.5 HoloLens 配件使用

HoloLens 配件主要有点击器和蓝牙键盘，这两个配件都可以通过蓝牙配对连接 HoloLens，从而进行一些输入。

1）HoloLens 点击器（HoloLens Clicker）是第一款专门为 HoloLens 特别定制的配件，HoloLens Clicker 允许用户以手部移动和点击来实现点击和滑动操作，可以用来替换 Air-tap（点击手势），但并不是所有的手势都可以通过 Clicker 完成。



配件的配对方式很简单，进入 HoloLens 设置中，打开蓝牙开关，按住点击器的配对按钮即可实现配对。

2）蓝牙键盘。在 HoloLens 应用中需要使用键盘的地方都可以通过蓝牙键盘进行输入。这里推荐 Microsoft Universal Foldable Keyboard 和 Microsoft Designer Bluetooth Desktop 这两种蓝牙键盘。

7.2 HoloLens 使用与开发环境配置

7.2.1 使用 Windows Device Portal

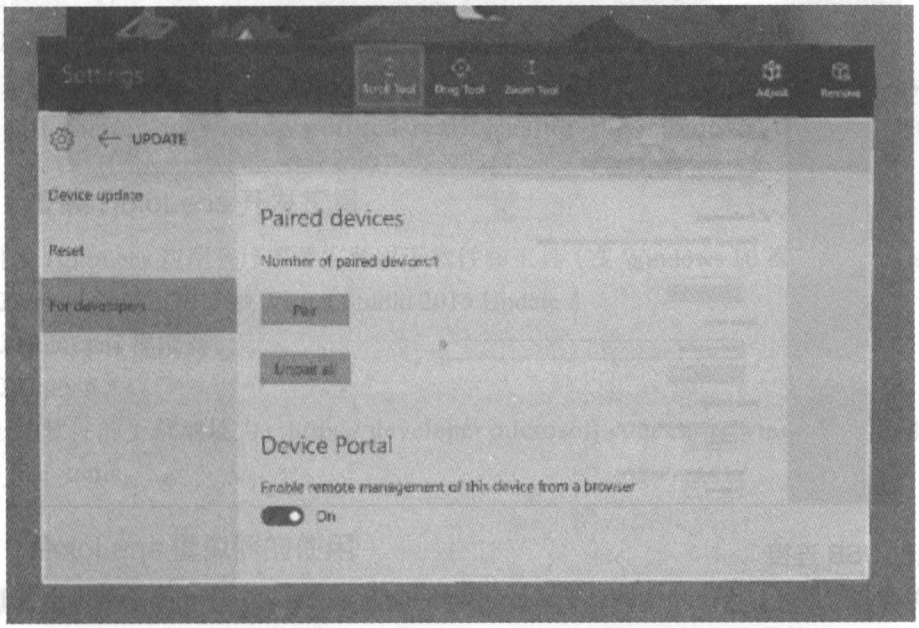
Windows Device Portal 可以让用户通过 Wi-Fi 或 USB 去配置或管理 HoloLens 设备，Windows Device Portal 这套系统其实是 HoloLens 上的一个 Web 服务器，因此我们可以通过计算机上的浏览器去访问，这套系统中有许多工具可以帮助我们来管理 HoloLens 设备。

本节主要讲解 Windows Device Portal 这套系统的功能和使用。

安装 Windows Device Portal

1）进入 HoloLens 设置中，依次选择 Update → For Developers 选项，并打开 Device Portal。

2）在设置中打开 Developer mode（开发者模式），如下图所示：



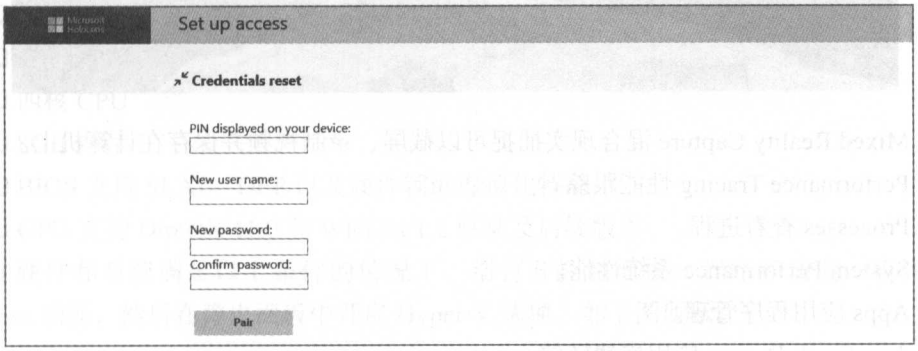
通过 Wi-Fi 连接

1）将 HoloLens 连接到中。

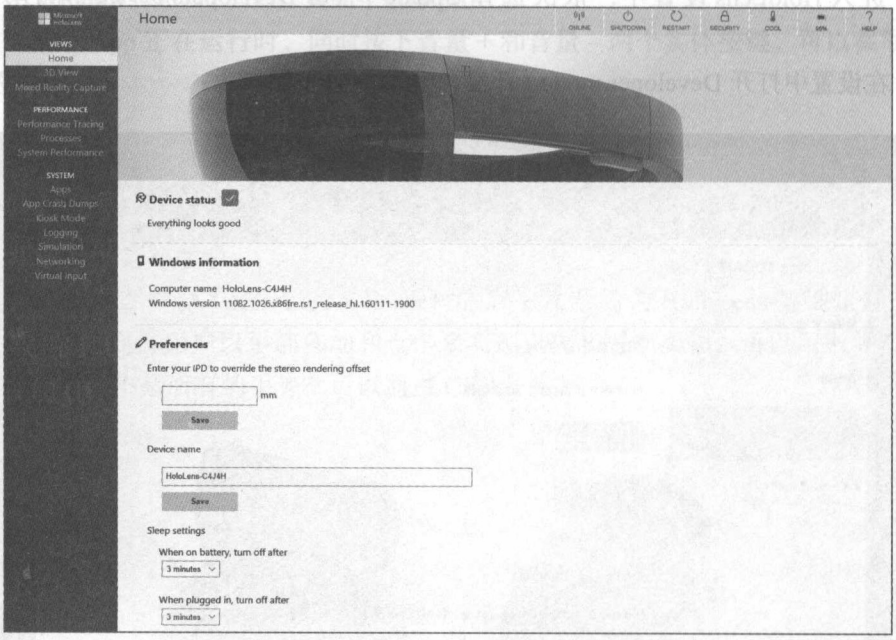
2）在 HoloLens 上查看设备 IP 地址，查看方式为 Settings → Network & Internet → Wi-Fi → Advanced Options.

3）打开浏览器并输入 https://<HoloLensIP 地址 >, 例如：https://192.168.1.2，如果浏览器提示不安全，也可继续访问。

4）如果进入 Set up access，则根据提示创建用户名、密码以及 HoloLens 设备上显示的 PIN 码，并点击 Pair 按钮进行配对。



5）最终进入 Windows Device Portal 窗口，如下图所示：



通过 USB 连接

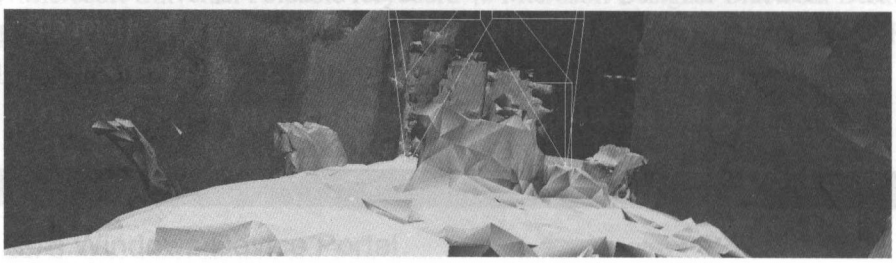
1）确保安装了“Visual Studio Update 1 with the Windows 10 developer tools”工具。

2）用 USB 数据线连接 PC 和 HoloLens 设备。

3）打开 PC 浏览器，输入 http://127.0.0.1:10080 即可。

通过 Wi-Fi 和 USB 都可以连接到 Windows Device Portal，在最终显示的窗口上，左边是 Windows Device Portal 提供的一系列工具，包括：

❑ 3D View 用来显示 HoloLens 周围的环境。



❑ Mixed Reality Capture 混合现实捕捉可以截屏、录制视频并保存在计算机上。

❑ Performance Tracing 性能跟踪。

❑ Processes 查看进程。

❑ System Performance 系统性能。

❑ Apps 应用程序管理。

❑ App Crash Dumps 应用崩溃转储。

- File Explorer 文件浏览。
- Kiosk Mode 该模式限制用户开启新的 App、bloom 手势和 Cortana 禁用。
- Logging 日志管理。
- Simulation 模拟功能，记录和回放输入数据进行测试。
- Networking 网络。
- Virtual Input 虚拟输入。

工具的详细使用方法可以查询 HoloLens 官方文档，链接如下：https://developer.microsoft.com/en-us/windows/mixed-reality/using_the_windows_device_portal。

7.2.2 安装 HoloLens 开发工具

开发 HoloLens 应用程序需要安装以下软件和工具（在 Windows 10 操作系统下）：

- Visual Studio 2017 或 Visual Studio 2015 Update 3
- HoloLens 模拟器
- Unity 5.5

上述软件的下载地址为：https://developer.microsoft.com/en-us/windows/mixed-reality/install_the_tools。

7.2.3 HoloLens 模拟器的使用

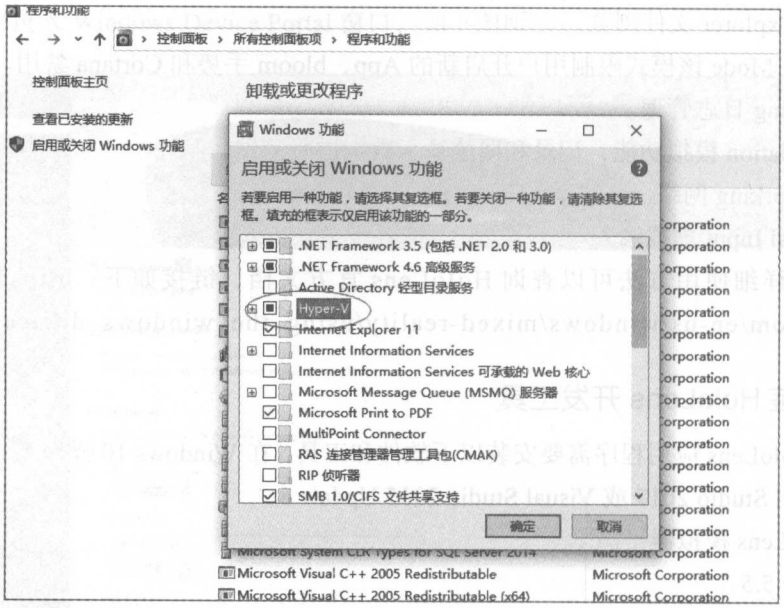
就目前 AR 技术相关的行业发展来看，HoloLens 这种具有革命意义的 AR 硬件产品，其昂贵的价格让大多数开发者望而却步，如果一个团队想开发基于 HoloLens 的产品，对于团队来说，人手一台 HoloLens 测试机无疑是不现实的，因此，HoloLens 官方提供了一款模拟器工具，HoloLens 模拟器可以让开发者脱离 HoloLens 真机在 PC 上测试全息应用，一些 HoloLens 中利用传感器的输入方式，在模拟器中用鼠标、键盘或者 Xbox 控制器进行模拟，当我们需要在真机上测试时，也不需要修改程序，这无疑是 HoloLens 开发者的福音。

可以从链接 https://developer.microsoft.com/en-us/windows/mixed-reality/install_the_tools 下载安装模拟器。

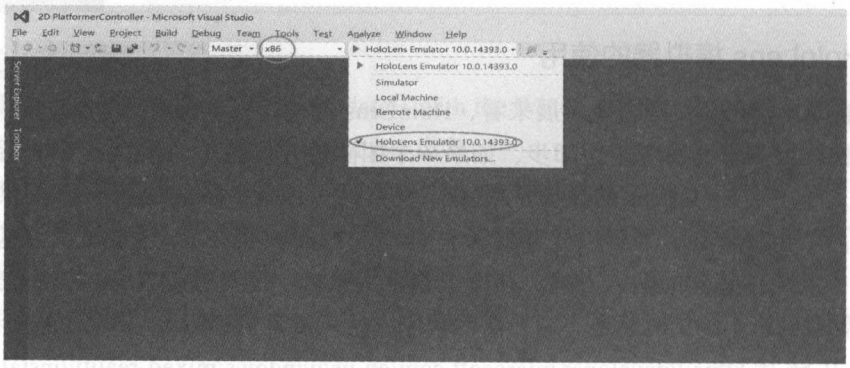
HoloLens 模拟器的安装，对系统有一定的要求，因此在开始安装之前，请先确认电脑和操作系统是否符合以下条件：

- 64 位 Windows 10 操作系统，企业版或教育版
- 64 位 CPU
- 四核 CPU
- 8GB 内存
- BIOS 支持 SLAT、DEP 以及硬件辅助虚拟化特征
- GPU 支持 DirectX 11.0 和 WDDM 1.2 驱动及后续版本

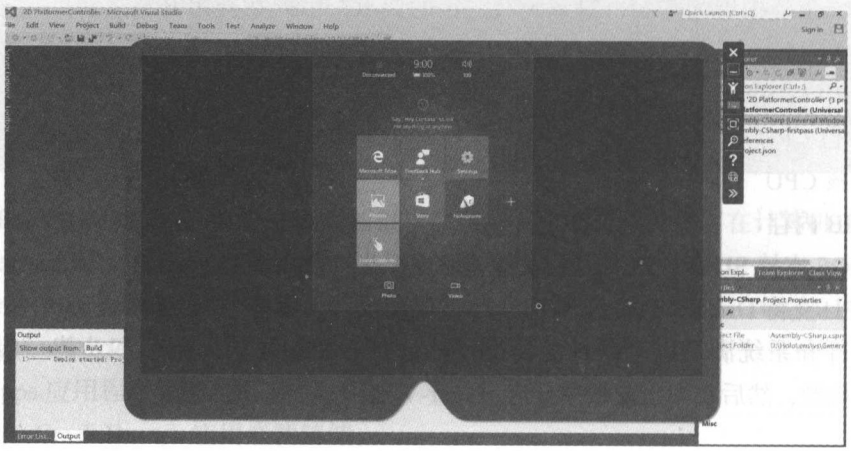
在硬件和系统满足以上条件的情况下，请打开控制面板→程序和功能→启用或关闭 Windows 功能，然后在弹出面板中开启 Hyper-V 选项，如下图所示。



设置完成后，我们打开一个已经导出到 Visual Studio 的工程，然后依照下图设置：



这样在运行时就会打开模拟器运行程序：



模拟器可以模拟 HoloLens 的使用，操作方式类似于 3D 游戏，基本的输入模拟用以下方式：

- 模拟移动：使用键盘 W、A、S、D 模拟用户在场景中移动
- 上下左右查看：点击并拖动鼠标，或使用键盘上下左右按键
- Air-tap 手势：鼠标右键点击或使用键盘 Enter 键
- Bloom 手势：键盘 Windows 按键或 F2
- 用手拖动：按下 Alt 键并按住鼠标右键拖动鼠标

上面是 HoloLens 模拟器的基本使用方法，当然，如果我们有 HoloLens 设备，直接使用设备调试是最好的方式。

7.3 使用 Unity 开发 HoloLens 全息应用

在开始开发之前，确保你已经按照 7.2 节中的内容完成了基础环境的搭建和测试。开发 HoloLens 全息应用最为便捷高效的方式是使用 Unity 引擎进行开发，Unity 也是 Microsoft 官方推荐的 HoloLens 开发工具，因此，熟练使用 Unity 是学习 HoloLens 开发的前提。

Unity 5.4 版本推出了 Unity HoloLens Technical Preview 版本，用来支持 HoloLens 的导出，当然，在 Unity 5.5 推出之后，HoloLens 开发所需模块已经内置到 Unity 5.5 版本中，因此，我们只需要安装 Unity 5.5 即可开发 HoloLens 应用。

7.3.1 配置适用于 HoloLens 开发的 Unity 工程

配置适用于 HoloLens 开发的 Unity 环境需要一系列操作，总体来说分为两部分：

场景（per-scene）的设置和工程（per-project）的设置。

场景设置

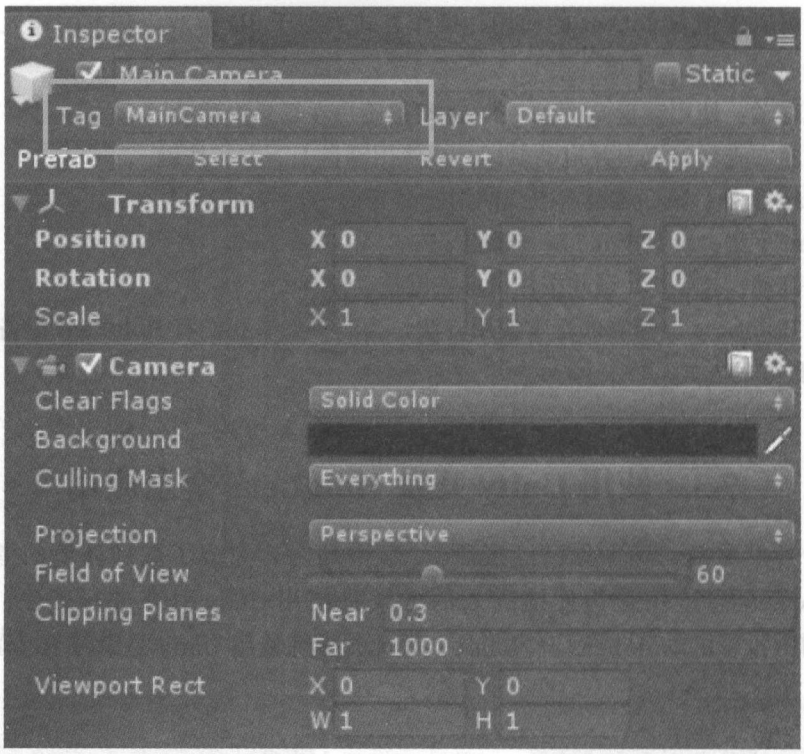
1）在新建的 Unity 场景中，将 Main Camera 的 position 设置为 (0,0,0)，这样的话，用户戴上 HoloLens 设备之后，头部就处于 Unity 世界的原点。

2）设置 Main Camera 的 Clear Flags 为 Solid Color 模式。

3）设置 Main Camera 的 Background 颜色值 RGBA 为（0,0,0,0），也就是将背景设置为黑色。这里设置为黑色的原因是 HoloLens 在渲染时认为黑色就是透明的，这样我们通过 HoloLens 的光学镜片，才能够看见现实中的物体。

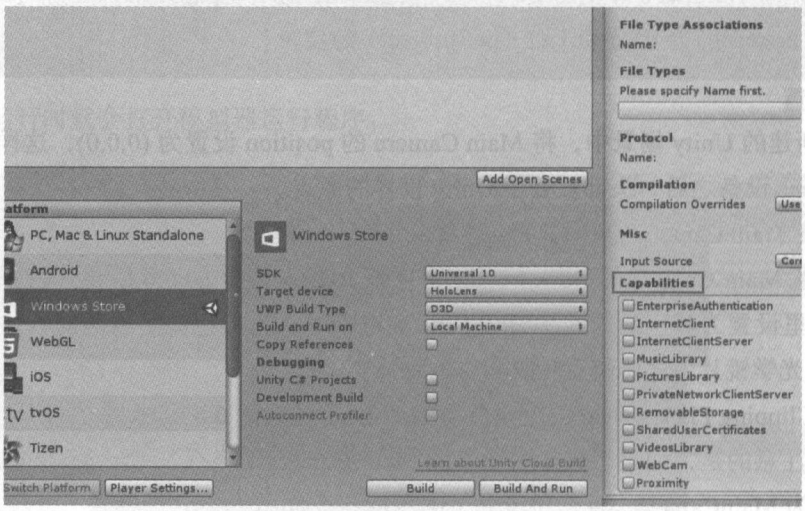
4）将 Clipping Planes-Near 的值设置为 HoloLens 建议的 0.85（米）。

这里要注意的是，如果将场景中自带的 Main Camera 删除了，那么新建的 Camera 物体一定要标记为 MainCamera，标记位置如下图所示：



工程设置

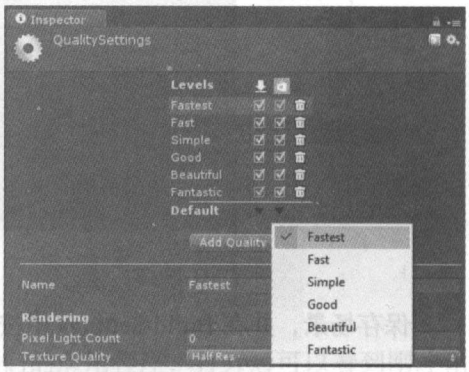
如果我们要开发 HoloLens 应用，根据不同应用上的功能需求，要在 Unity 中将对应的特性在 manifest 文件中声明，在 Unity 中，我们可以在 Player Settings → Windows Store → Publishing Settings → Capabilities 属性下进行设置，如下图所示：



下表列举了常用的功能选项和功能需求的对应关系。

功能选项	功能需求
WebCam	截屏和录屏功能
SpatialPerception	SurfaceObserver 和空间锚
Microphone	录屏，听写识别，键盘识别等
PicturesLibrary / VideosLibrary / MusicLibrary	截屏和录屏
InternetClient	听写识别

同时我们还需制定 Unity 导出为 Universal Windows Platform 平台，将 Windows Store 选项中的 SDK 设置为 Universal 10，Build Type 设置为 D3D。



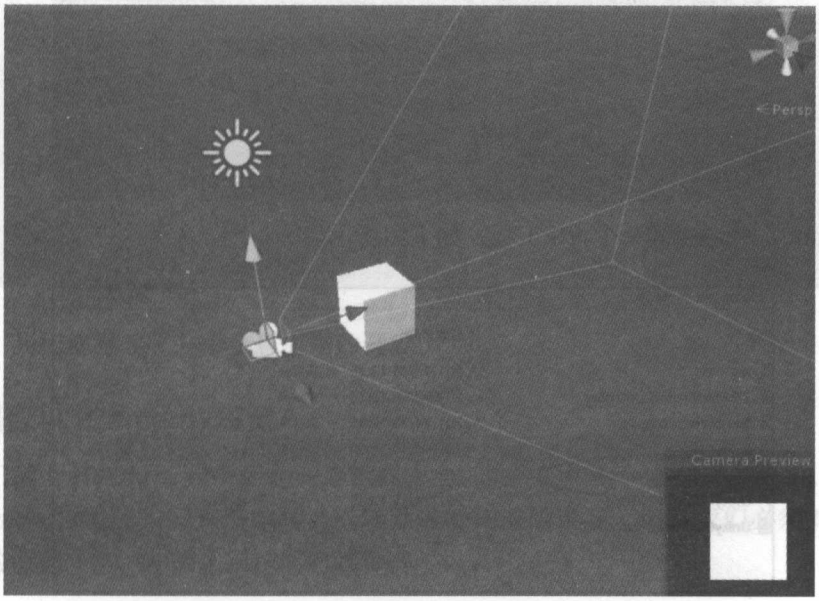
为了保证在 HoloLens 中高效渲染，我们必须对 Unity 渲染的质量进行限制，在 Project → Quality 选项中，将质量设置为 Fastest, 如右图所示。

最终，我们需要让 Unity 知道导出的应用是一个全息视图而不是 2D 视图，因此要将 HoloLens 设置为 virtual reality device（虚拟现实设备）。

设置方式为，Player Settings → Windows Store → Other Settings，在 Renderer 栏目中，勾选 Virtual Reality Supported 选项。

接下来，通过一个简单的案例来开始我们的第一个全息应用。

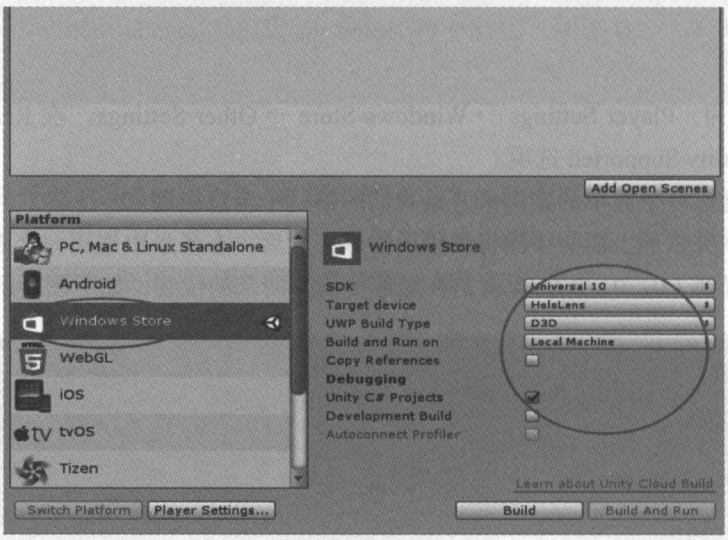
新建工程，并依照上面的步骤将工程设置好，然后，在场景中创建一个立方体，并将立方体的位置设置到（0，0，2），也就是在摄像机前方两米的位置，如下图所示。



新建一个 cube.cs 添加到立方体上，然后在其中添加如下代码：

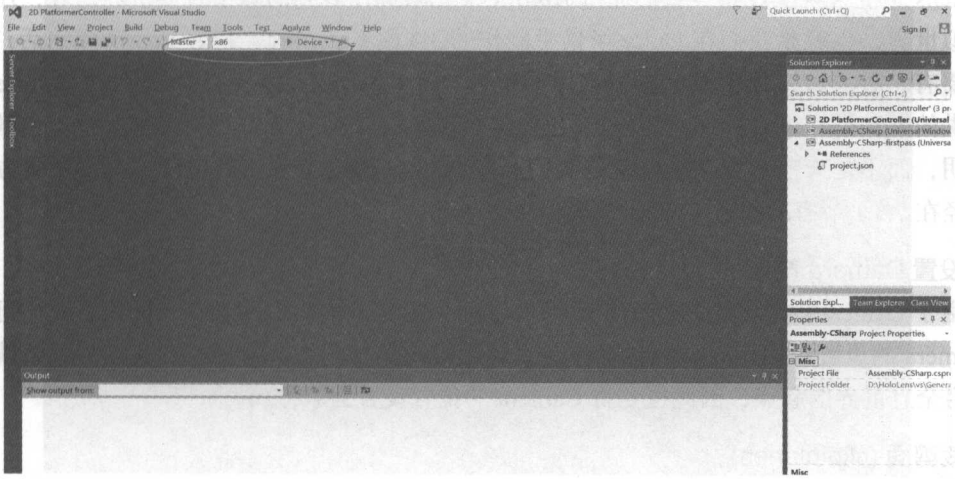
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class cube : MonoBehaviour
{
    void Start ()
     {

    }

    void Update ()
     {
         //让立方体绕 Y 轴旋转
         transform.Rotate(new Vector3(0, 1, 0));
    }
}
```

保存场景，并在 Build Settings 中按照下图进行设置：

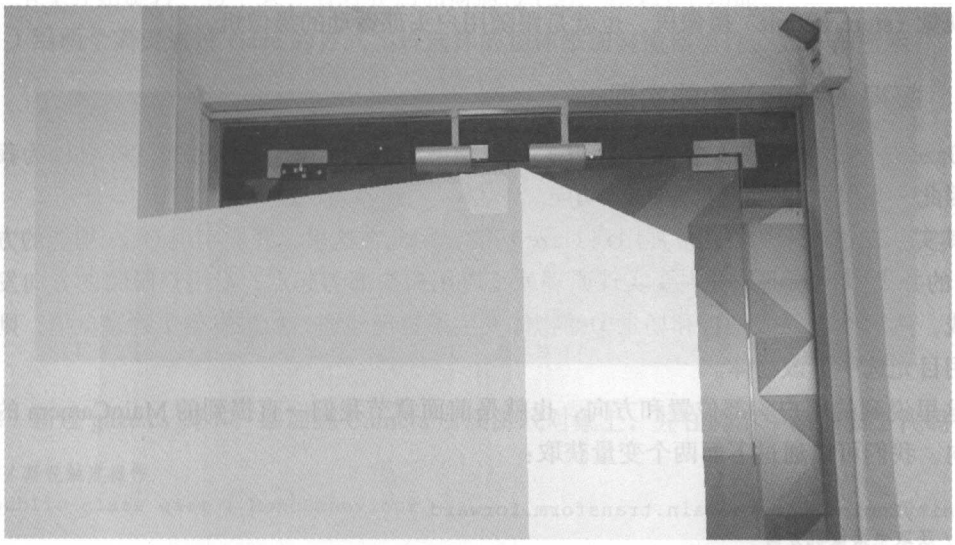


点击 build，将会生成一个 VS 工程，如下图所示。

双击图中的 .sln 文件，就会在 VS 中打开工程，然后使用 USB 连接 HoloLens 设备，并在 HoloLens 中打开开发者模式，在 VS 中依照下图进行设置：



点击运行，完成部署后，进入 HoloLens 找到我们的应用程序，打开后可以看到刚才创建的旋转立方体。



这样，我们就完成了一个全息应用的开发。

7.3.2 摄像机 (Camera) 设置

当我们戴上 HoloLens 设备后，头部就成为全息世界的中心，在全息应用中，Unity 场景里面的 Camera 会跟随我们头部的移动或转动进行相应的改变。但是我们必须对 Camera 进行一系列的设置操作，才能达到预期的效果。

混合现实渲染 (mixed reality rendering) 设置

Unity 场景中的 Camera，默认以一个天空盒作为背景，并不会有真实环境的显示，但是 HoloLens 是混合现实设备，我们通过 HoloLens 光学镜片看到的除了真实世界之外，还应当有虚拟世界，也就是我们在 Unity 场景中制作的 3D 模型、动画、特效等。这样的话，我们就需要有一种方式，让不显示虚拟物体的地方，都能够显示背后的真实环境，HoloLens 官方提供的做法是，将 Camera 的背景颜色设置为黑色，HoloLens 渲染时，如果是黑色则处理成透明，而不是天空盒背景，这就是我们开发 HoloLens 时摄像机的机理，具体设置方式其实已经在 7.3.1 节有所提及，故不再赘述。

设置 Camera 位置

由于 Camera 的位置会跟随用户头部的移动和旋转做出相应的动作，因此，我们可以认为 Camera 的位置跟用户眼睛的位置是一致的，当我们打开一个应用时，我们的眼睛就是我们开启全息世界的起点，因此也要将 Camera 的位置设置为 (0,0,0)。

裁剪面 (clip planes)

根据 HoloLens 官方建议，我们需要将 Camera 的近裁剪面（Near）设置为 0.3 到 0.85 之间。

多摄像机

当场景中有多个 Camera 存在时，Unity 会自动将标记为 MainCamera 的摄像机作为双目立体视觉（stereoscopic）摄像机，也就是跟随用户头部运动的摄像机。

7.3.3 凝视（Gaze）功能实现

Gaze 是 HoloLens 中的主要交互方式，但是可惜 Unity 中并无自带的 API 可以为我们所用，因此，我们只能自己实现 Gaze 功能。

其实，在 VR 开发中也常常用到 Gaze 功能，因此在这里，我们实现 Gaze 功能的方式与 VR 中的并无多大区别，其主要实现原理就是以用户头部为起点，沿用户眼睛的方向发射一条射线，然后判断射线是否和场景中的物体发生了物理碰撞，如果发生了物理碰撞，则认为我们用目光选中了该物体。

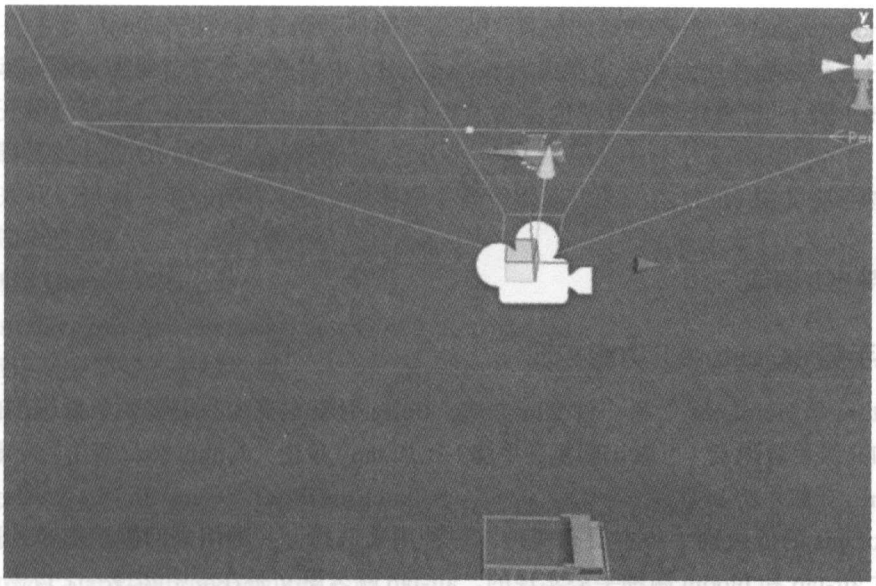
这里说到的用户头部位置和方向，也就是前面章节我们一直提到的 MainCamera 的位置和方向，我们可以通过下面两个变量获取：

```
UnityEngine.Camera.main.transform.forward
// 获取主摄像机方向
UnityEngine.Camera.main.transform.position
// 获取主摄像机位置
```

而判断碰撞可以调用 Unity 中的 Physics.RayCast 函数，根据函数返回结果判断是否选中物体。

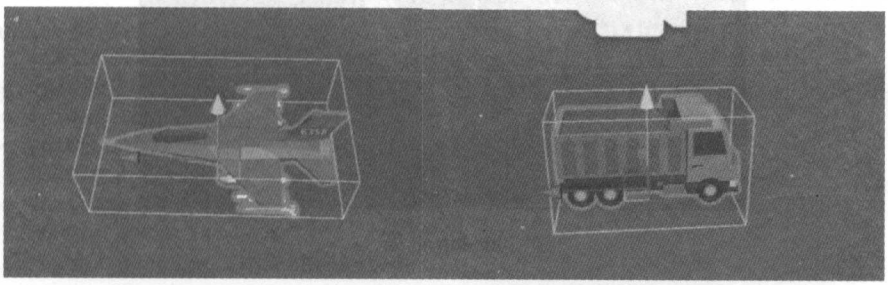
接下来，我们通过一个简单示例来实现 Gaze 的功能。

参考 7.3.1 节和 7.3.2 节内容设置工程，我们在场景中添加两个 3D 物体，如下图所示。



我们将一架飞机的模型和一辆卡车的模型分别放在摄像机的前方和后方，飞机和卡车距离摄像机的位置都为 2 米，我们希望用户看到飞机的时候飞机开始旋转，转身看到身后的卡车时，卡车开始旋转，接下来，看看我们如何通过 Gaze 实现这个功能。

1）给两个需要通过 Gaze 的方式实现选择的物体添加碰撞器组件，如下图所示：



2）新建 gaze.cs 脚本，添加到 Camera 所属游戏对象上，并在脚本中添加如下代码：

```
//凝视触发操作
public class gaze : MonoBehaviour
{
    void Start ()
     {
    }

    void Update ()
     {
         // 通过 Physics.Raycast 方法检测是否与某物体有碰撞
         RaycastHit hitInfo;
         if (Physics.Raycast(Camera.main.transform.position,
        Camera.main.transform.forward,
```

```
            out hitInfo, 20f, Physics.DefaultRaycastLayers))
        {
            Transform tran = hitInfo.transform;
            // 控制物体旋转
            tran.Rotate(new Vector3(0, 1, 0));
        }
    }
}
```

3）导出并测试。

7.3.4 手势（Gesture）功能实现

Gesture 是 HoloLens 中另一种交互方式，Unity 引擎本身为我们提供了低级别的手势数据访问和高级手势识别（可以识别复杂手势）比如 tap（单击）、double tap（双击）、hold（按住）、navigation（导航手势）等。

HoloLens 的低级别手势主要有两种，一种用来选择，一种用来回到主菜单，这两个功能是通过 air-tap 和 bloom 两种手势实现的。air-tap 顾名思义是在空中实现一个点击动作，而 bloom 手势是鲜花绽开的意思，这两种手势的操作方式如下图所示：

air-tap 手势

bloom 手势

高级别手势 Hold 有几秒钟的持续时间，类似于鼠标的按住事件，Manipulation 手势可以用来移动，缩放或旋转某个全息物体，Navigation 手势类似于游戏中的虚拟操纵杆，通过点击手势开始，然后在点击处为中心的标准立方体范围内移动手部，获得 -1 到 1 的坐标变化，来控制场景中的 UI 或游戏物体等。

接下来我们讲解一下如何在 Unity 中实现手势识别的功能，在 Unity 中实现手势识别主要有以下四个步骤：

1）创建 Gesture Recognizer。

GestureRecognizer recognizer = new GestureRecognizer();

2）指定需要捕捉的手势类型。

recognizer.SetRecognizableGestures(GestureSettings.Tap);

3）捕捉到手势后的处理。

recognizer.TappedEvent += TapEventHandler;

4）开始捕捉手势。

recognizer.StartCapturingGestures();

5）结束手势捕捉。

recognizer.StopCapturingGestures();

通过上述步骤就能够实现手势捕捉的功能，我们通过一个简单的案例来讲解整个实现过程。

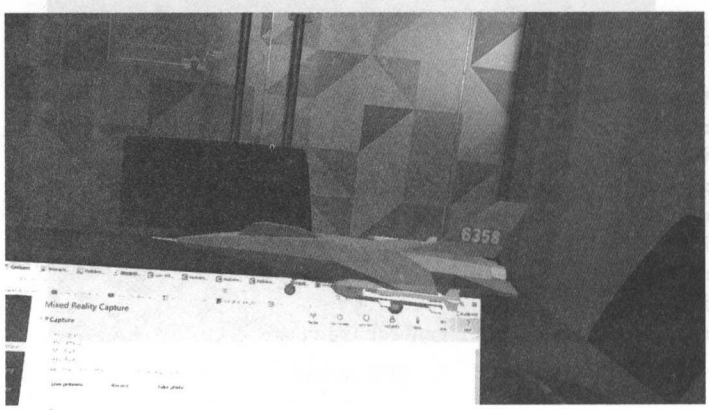
我们依然选择 7.3.3 节中的场景，给场景中的物体添加 Collider 碰撞器，接下来，创建一个 gesture.cs 脚本，并作为组件添加在 Camera 物体上，脚本中的代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//需引用以下命名空间
using UnityEngine.VR.WSA.Input;

public class gesture : MonoBehaviour
{
    //定义一个手势识别器
    GestureRecognizer recognizer;

    void Start ()
    {
        //初始化识别器
        recognizer = new GestureRecognizer();
        //设置识别类型，这里为识别单击和双击
        recognizer.SetRecognizableGestures(GestureSettings.Tap |
        GestureSettings.DoubleTap);
        //注册手势识别事件
        recognizer.TappedEvent += TapEventHandler;
        //开启手势捕捉功能
        recognizer.StartCapturingGestures();
    }
```

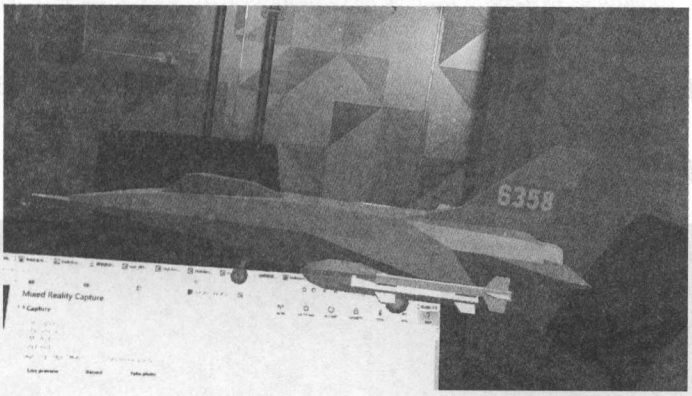
```
    //手势识别事件
    //InteractionSourceKind为事件的来源，是一个枚举值，有以下四个取值
    //Other: 其他
    //Hand: 手
    //Voice: 语音
    //Controller: 控制器
    //tapCount表示点击次数，如果tapCount = 1则表示单击，tapCount=2表示双击
    //headRay表示当前从用户头部前方发射的射线，可以用来检测选取物体
    void TapEventHandler(InteractionSourceKind source, int tapCount, Ray headRay)
    {
        //进入该函数表示已经捕捉到手势
        if (source == InteractionSourceKind.Hand)
        {
            //通过Physics.Raycast方法检测头部射线是否与场景物体有碰撞
            //简单来说，就是用户是不是盯着某个3D物体，如果盯着，则放大，否则不放大
            RaycastHit hitInfo;
            if (Physics.Raycast(headRay, out hitInfo, Mathf.Infinity))
            {
                Transform tran = hitInfo.transform;
                //点击后放大物体
                tran.localScale += new Vector3(0.01f, 0.01f, 0.01f);
            }
        }
    }

    void OnDestroy()
    {
        //停止捕捉手势
        recognizer.StopCapturingGestures();
        //取消手势事件
        recognizer.TappedEvent -= TapEventHandler;
    }


}
```

导出到HoloLens运行之后的效果如下。

进入应用时飞机的大小：



手势点击后飞机变大：



通过上面的案例，我们学习了如何在 HoloLens 中实现手势交互的功能，除了点击之外，HoloLens 还有其他手势，因篇幅所限，在此不一一列举，有兴趣的读者可以自行查阅相关文档和资料。

7.3.5 语音输入（Voice input）功能实现

语音输入是 HoloLens 中继 Gaze 和 Gesture 之后的第三种交互方式，在 Unity 中，我们可以通过以下三种方式实现语音输入。

- KeywordRecognizer（关键字识别），提供一个 string 类型的关键字数组，用户在读出数组对应单词后将会识别。
- GrammarRecognizer（语法识别），通过指定一个 SRGS 语法规范文件定义识别。
- 上面两种识别方式统称为 PhraseRecognizer（短语识别）。
- DictationRecognizer（听写识别），用户可以说出任何单词，识别后可以用来显示用户说话的内容。

语音输入目前并不支持中文，所以基本上都是针对英语进行开发，另外需要注意的一点是，DictationRecognizer（听写识别）和 PhraseRecognizer（短语识别）在同一时刻只能使用一种，不能同时开启两种识别功能。

接下来，我们通过关键字识别来掌握 HoloLens 语音输入的实现。

1）我们需要引用必要的命名空间 UnityEngine.Windows.Speech。

2）定义关键字识别器。

```
KeywordRecognizer recognizer;
```

3）定义识别关键字。

```
Dictionary<string, Action> keywords = new Dictionary<string, Action>();
```

4）注册识别事件。

```
recognizer.OnPhraseRecognized += OnPhraseRecognized;
```

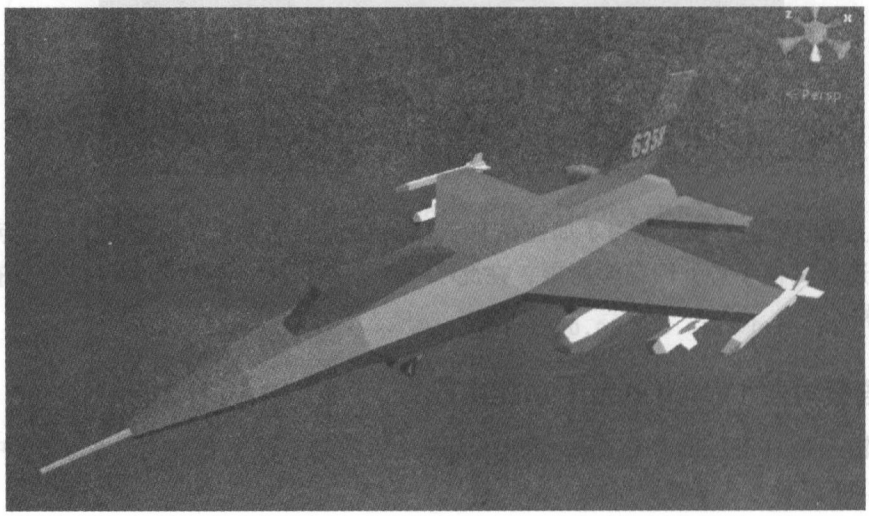
5）开启识别功能。

recognizer.Start();

上面是关键字识别的主要步骤，接下来通过案例的方式实现关键字识别功能。

我们依然采用前面章节的场景，该场景中的飞机模型下有子物体导弹，默认导弹和飞机是在一起的，在运行中，如果用户说出“Fire”，程序会识别“Fire”关键词，并发射导弹。

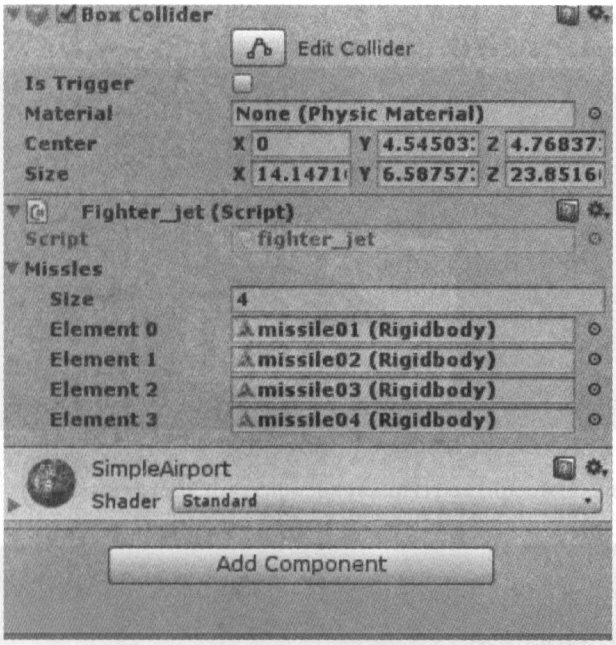
飞机与导弹模型如下图所示：



我们创建一个 fighter_jet.cs 脚本并添加到飞机模型上，具体代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class fighter_jet : MonoBehaviour
{
    //所有的导弹子物体
    public Rigidbody[] missles;
    //记录发射导弹的 ID
    int fireIndex = 0;

    //通过给导弹子物体一个向前的速度实现发射导弹功能
    public void Fire()
    {
        if(fireIndex < missles.Length)
        {
            missles[fireIndex].velocity = missles[fireIndex].transform.forward;
            fireIndex++;
        }
    }
}
```

将该脚本添加到飞机模型上后，将导弹子物体赋值到对应数组上，如下图所示：

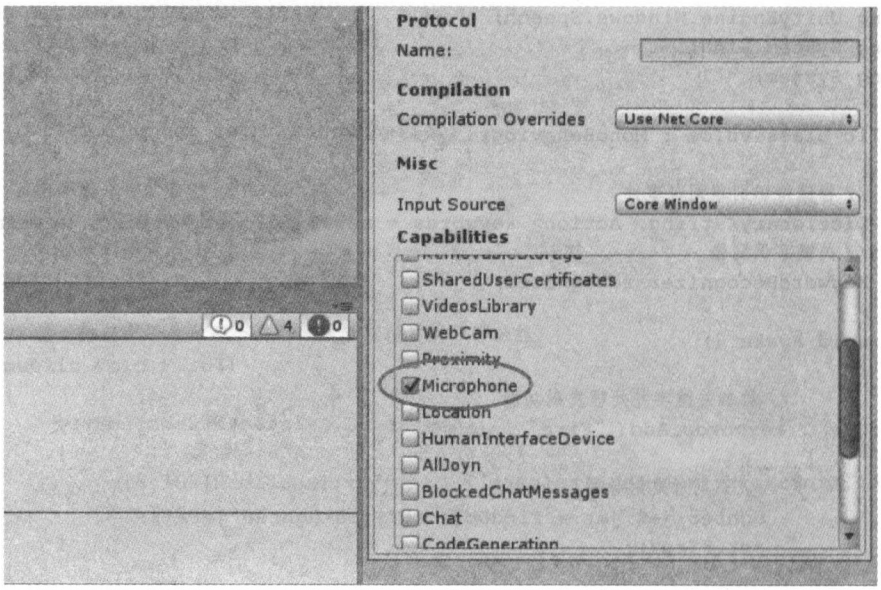


飞机的发射功能处理完之后，再创建一个 voice.cs 脚本并添加到 Camera 物体上，voice.cs 脚本中的代码如下：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//引用命名空间
using UnityEngine.Windows.Speech;
using System.Linq;
using System;

public class voice : MonoBehaviour
{
    //存储待识别单词与事件
    Dictionary<string, Action> keywords = new Dictionary<string, Action>();
    //关键字识别器
    KeywordRecognizer recognizer;

    void Awake ()
    {
        //添加关键字并处理发射功能
        keywords.Add("Fire", () =>
    {
        //识别后发射导弹
        fighter_jet jet = FindObjectOfType<fighter_jet>();
        jet.Fire();
    });

        //创建识别器
        recognizer = new KeywordRecognizer(keywords.Keys.ToArray());
```

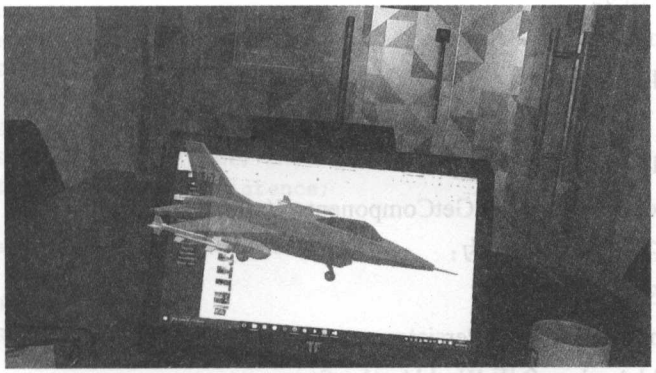
```
        //开启识别
        recognizer.Start();
    }

    //事件监听
    private void OnEnable()
    {
        recognizer.OnPhraseRecognized += OnPhraseRecognized;
    }

    //取消监听
    private void OnDisable()
    {
        recognizer.OnPhraseRecognized -= OnPhraseRecognized;
    }

    void OnPhraseRecognized(PhraseRecognizedEventArgs args)
    {
        Action keywordAction;
        if (keywords.TryGetValue(args.text, out keywordAction))
        {
            //调用关键字识别事件
            keywordAction.Invoke();
        }
    }
}
```

代码完成之后，我们需要通过 Unity Player Settings → Publishing Settings → Capabilities 打开 Microphone 选项，如下图所示。



导出运行之后，发出“Fire”指令，可以看到导弹从飞机机身发射向前飞走，如下图所示。

发射前：



发射后：

7.3.6 世界锚（World Anchor）与场景保持（Persistence）功能实现

在开始本节内容之前，首先需要明确两个概念：real world 和 unity world。当我们打开 HoloLens 的某个全息应用时，将会产生两个世界，其中一个是我们看到的真实世界，即 real world，另一个就是由全息应用创建的虚拟世界，因为我们这里采用 unity 进行开发，所以也称之为 unity world。HoloLens 给我们带来的是混合现实（Mixed Reality）体验，也就是将 real world 和 unity world 进行了混合。

HoloLens 中的混合现实体验不仅仅是让我们能同时看到真实与虚拟世界，其核心在于，我们能够通过程序控制让虚拟物体与真实世界产生有机结合，这是 HoloLens 强大的地方，也是我们最值得去探究的奥秘。

而世界锚（World Anchor），就是这种真实与虚拟有机结合的桥梁，World Anchor 能够让我们将一个 Unity 中的 GameObject 定位到真实世界中的某个位置并保持它的旋转状态。

场景保持（Persistence）与世界锚相互结合，能够将物体定位在真实世界中并保存下来。举例来说：当打开一个全息应用时，将场景中某个物体移动到特定位置，然后添加世界锚并保存下来，关闭应用，再次进入时，物体会恢复到上次保存的位置。

World Anchor 实现的主要内容为：

1）引用命名空间。

```
using UnityEngine.VR.WSA;
```

2）添加 World Anchor。

```
WorldAnchor anchor = fighter_jet.AddComponent<WorldAnchor>();
```

3）删除 World Anchor。

```
DestroyImmediate(fighter_jet.GetComponent<WorldAnchor>());
```

Persistence 实现的主要内容为：

1）引用命名空间。

```
using UnityEngine.VR.WSA.Persistence;
```

2）加载 World Anchor 仓库 WorldAnchorStore。

```
WorldAnchorStore.GetAsync(OnStoreLoad);
```

3）保存 World Anchor。

```
store.Save("fighter_jet", anchor);
```

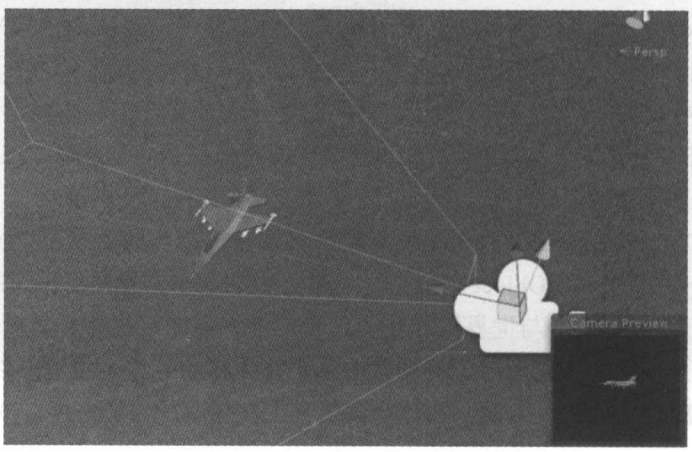
4）加载 World Anchor。

```
this.store.Load("fighter_jet", fighter_jet);
```

为了让大家更好地理解 World Anchor 和 Persistence 的特性，我们依然采用一个案例来说明这两大功能。由于此案例比前面章节的案例复杂，因此先对功能做一个描述。

在打开应用后，场景中的飞机模型处在 unity world 中的固定位置，用户用双击事件，让飞机模型跟随用户的头部进行移动，也就是不论用户在哪里，飞机总会跟随用户，再次双击之后，飞机就会被 World Anchor 锁定在 real world 当前位置并保存在 World Anchor Store 中，这时用户用 bloom 手势退出程序，再次进入应用时，可以看到飞机的位置保持在上次关闭之前的位置。

好了，在理解了应用功能之后，我们开始实现案例。首先，我们的场景依然采用之前的飞机模型，在场景中配置 Camera，接下来将飞机模型放在 Camera 前方（0，0，2）的位置，如下图所示：

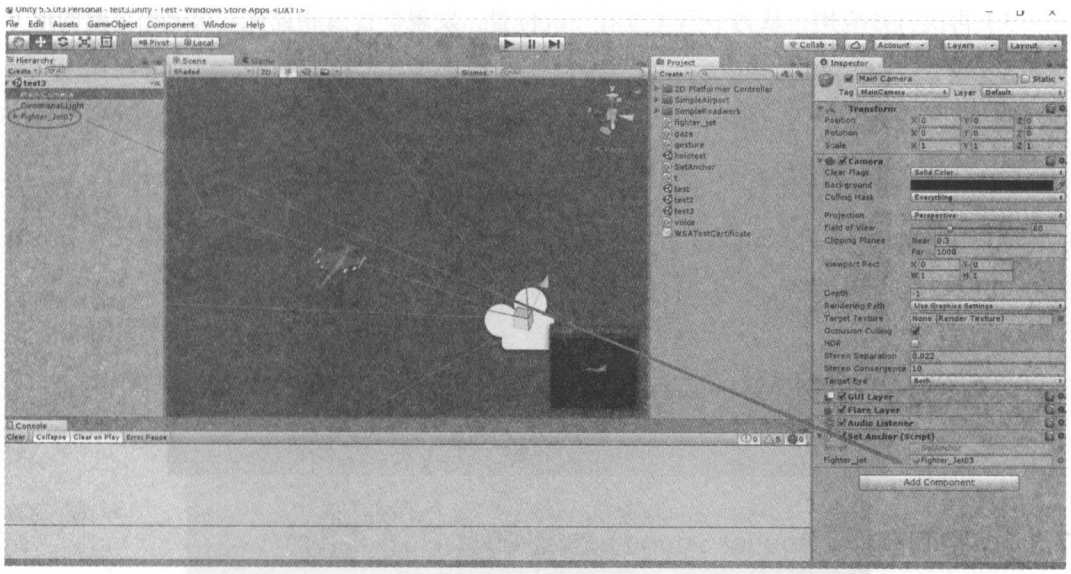


创建 SetAnchor.cs 脚本并添加如下代码：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
// 引入以下三个命名空间
using UnityEngine.VR.WSA;
using UnityEngine.VR.WSA.Input;
using UnityEngine.VR.WSA.Persistence;

public class SetAnchor : MonoBehaviour
{
    //场景中的飞机
    public GameObject fighter_jet;

    //手势识别，用来触发事件
    GestureRecognizer recognizer;
    //表示飞机是否跟随 Camera 移动和旋转
    bool followCamera = false;
    //World Anchor 仓库，用来存储和加载 World Anchor
    WorldAnchorStore store;

    void Start()
    {
        //初始化手势捕捉相关功能
        recognizer = new GestureRecognizer();
        recognizer.SetRecognizableGestures(GestureSettings.Tap |
        GestureSettings.DoubleTap);
        recognizer.TappedEvent += TapEventHandler;
        recognizer.StartCapturingGestures();
        //开始时飞机不跟随摄像机
        followCamera = false;
        //异步加载 World Anchor 仓库
        WorldAnchorStore.GetAsync(OnStoreLoad);
    }

    //World Anchor 加载完成回调函数
    void OnStoreLoad(WorldAnchorStore store)
    {
        //将加载后的 WorldAnchorStore 保存在变量 this.store 中
        this.store = store;

        //遍历查找已经存储的 World Anchor
        string[] ids = this.store.GetAllIds();
        for(int i = 0; i < ids.Length; i++)
        {
            //如果找到飞机的 World Anchor 信息
            if (ids[i] == "fighter_jet")
            {
                //将保存的 World Anchor 加载并应用到飞机模型上
                this.store.Load("fighter_jet", fighter_jet);
                break;
            }
        }
```

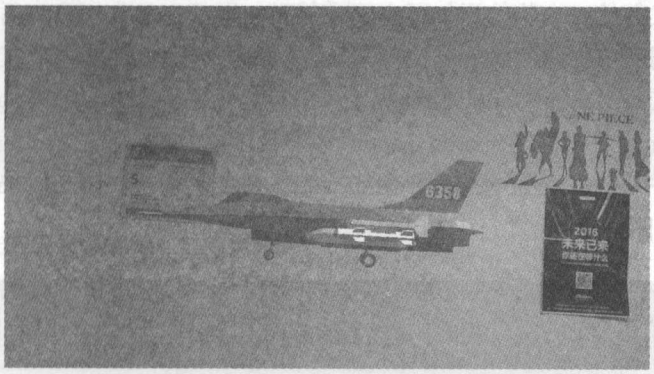
```
    }

    // 双击手势事件触发
    void TapEventHandler(InteractionSourceKind source, int tapCount, Ray headRay)
    {
        // 是否为双击事件
        if (source == InteractionSourceKind.Hand && tapCount == 2)
        {
            // 当飞机不跟随摄像机时，双击会让飞机跟随摄像机移动，用来后面固定到用户想要的位置
            if (followCamera == false)
            {
                // 删除已有 World Anchor
                WorldAnchor oldAnchor = fighter_jet.GetComponent<WorldAnchor>();
                if (oldAnchor != null)
                {
                    DestroyImmediate(oldAnchor);
                }
                // 设置飞机模型与摄像机相对位置并跟随相机
                fighter_jet.transform.localPosition = new Vector3(0, 0, 2);
                fighter_jet.transform.parent = Camera.main.transform;
                followCamera = true;
            }
                // 在飞机模型跟随摄像机时，双击会将飞机锁定在 World Anchor 上，并保存在
                WorldAnchorStore 中
            else
            {
                // 取消跟随
                fighter_jet.transform.parent = Camera.main.transform.parent;
                if(this.store != null)
                {
                    // 添加 World Anchor
                    WorldAnchor anchor = fighter_jet.AddComponent<WorldAnchor>();
                    // 保存 World Anchor
                    store.Save("fighter_jet", anchor);
                }
                followCamera = false;
            }
        }
    }


    void OnDestroy()
    {
        // 停止捕捉手势
        recognizer.StopCapturingGestures();
        // 取消手势事件
        recognizer.TappedEvent -= TapEventHandler;
    }
}
```

代码添加完成后，将脚本添加在 Camera 上，并将飞机的游戏物体赋值到脚本上，如下图所示：

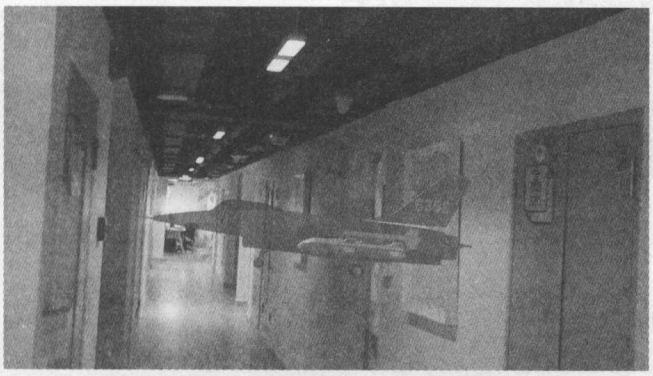


保存场景，导出并在 HoloLens 中运行得到以下效果：

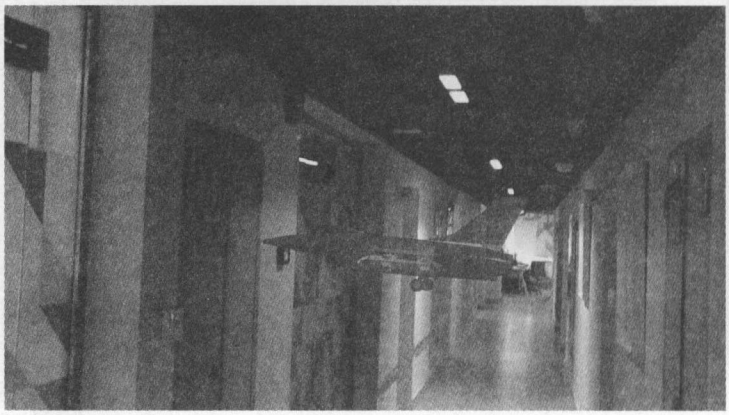
1）第一次打开时飞机的位置。



2）固定到以下位置后，退出应用。



3）再次打开应用，从另一角度查看飞机位置，发现位置被放置在上次退出之前的地方。

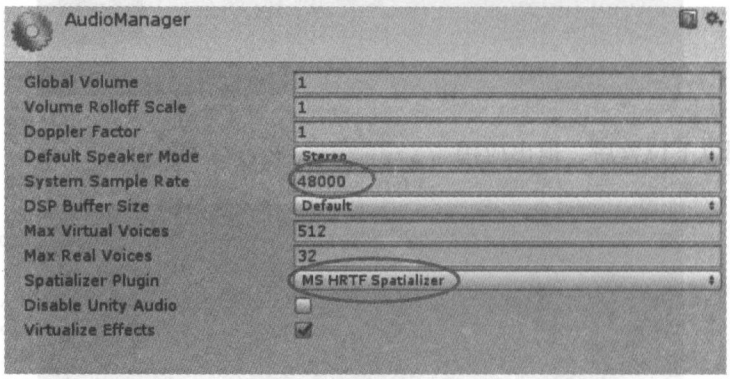


7.3.7 空间音效（Spatial Sound）功能实现

空间音效是 HoloLens 混合现实体验的又一重要特点，在 HoloLens 应用中，我们能看到的全息影像都是在我们凝视的方向，我们左侧的、右侧的、身后的这些全息影像到底在哪里，它们在做什么，在我们的视线离开它们的时候就无从得知了。而空间音效能够完美地解决这个问题，当我们给一个物体添加了空间音效时，即使它身处用户身后，但是当它发出声音时，我们可以判断出物体此时的方位，以及它与我们的距离，就跟真实世界一样。

那么，如何实现 Spatial Sound 功能呢？本节就为大家介绍 Spatial Sound 的实现。

首先，在 Unity 中开启 Spatial Sound。开启的方法很简单，通过 Edit → Project Settings → Audio → Spatializer 来开启 Microsoft HRTF，并将 System Sample Rate 设置为 48000 即可，如下图所示：



接下来，我们在前一节的飞机模型上添加一个 Audio Source 组件，按照如下步骤设置 Audio Source 属性：

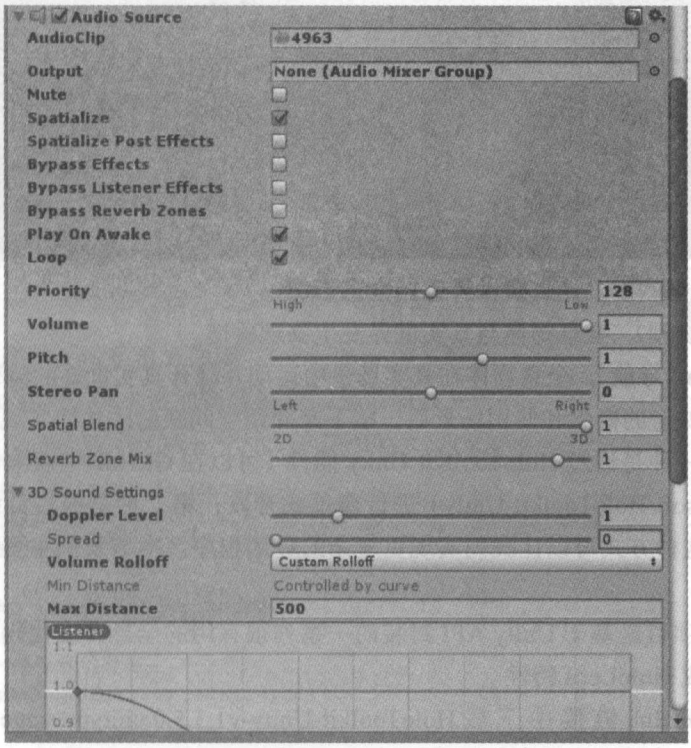
1）选中 Spatialize 属性。

2）设置 Spatial Blend 模式为 3D。

3）展开 3D Sound Settings，并将 Volume Rolloff 值设置为 Custom Rolloff。

4）为 Audio Source 添加一个音效片段。

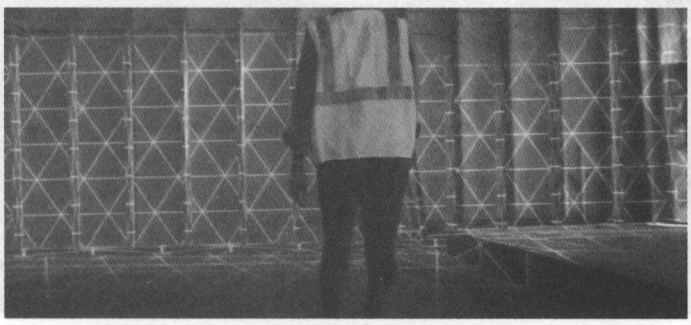
最终效果如下：



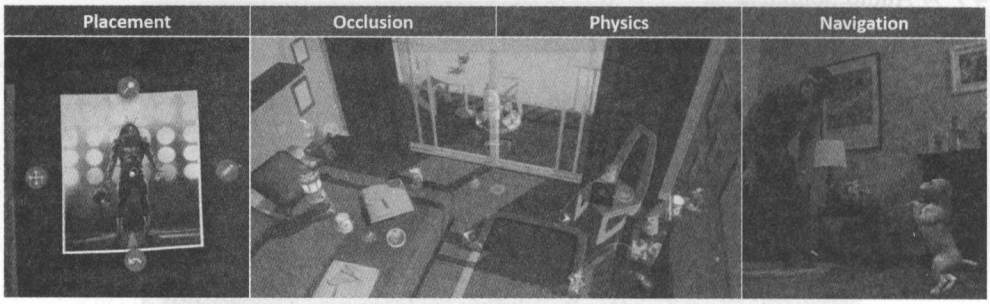
最后保存场景并在 HoloLens 上测试效果。

7.3.8 空间映射（Spatial Mapping）功能实现

本节内容主要为大家讲解 HoloLens 中空间映射的功能，空间映射是 HoloLens 中最重要的特性，它能够让虚拟物体与真实物体产生交互，比如让虚拟世界的篮球在真实世界的桌面上弹跳。它能将周围的环境扫描后生成 Mesh，对于用户来说，它就像魔法一样，让我们体验到神奇混合现实加全息应用，下图就是空间映射的效果。



空间映射功能有以下常见的应用场景：



- Placement（放置），比如将全息画贴在墙上。
- Occlusion（遮挡），让真实世界遮挡全息影像。
- Physics（物理），全息物体与真实世界发生物理碰撞。
- Navigation（导航），全息物体在真实场景中运动并绕开真实世界的障碍物。

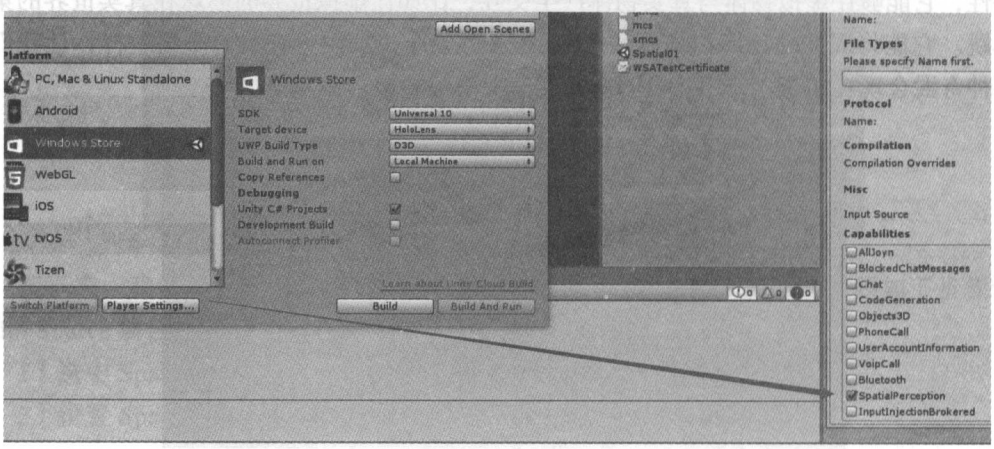
了解了空间映射的概念和使用场景后，接下来介绍空间映射的功能实现。实现空间映射有两种方式，第一种是通过 HoloToolKit Unity 插件（可以在 GitHub 中获取，链接为 https://github.com/Microsoft/HoloToolkit-Unity）进行傻瓜式开发；第二种是通过低级别空间映射的 API 进行开发，这种方式可以让我们实现更复杂的应用程序。本节我们主要通过第一种方式为读者讲解。

HoloToolkit 项目是基于 Unity API 封装的一系列很有用的全息开发代码工具集合，能帮助开发者快速集成 HoloLens 特性。

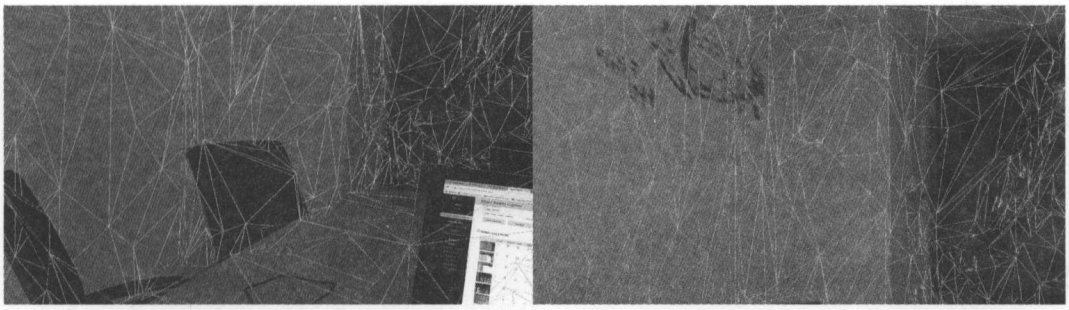
从前面的 GitHub 链接中下载 HoloToolkit-Unity-v1.5.5.0.unitypackage 插件包，新建 Unity 工程，并导入插件包。

新建场景，然后将场景中自带的 MainCamera 删除，找到 HoloLensCamera.prefab 预制件拖入场景中，接下来再找到 SpatialMapping.prefab 预制件拖入场景中。

保存场景，并通过 Build Settings → Player Settings → Publishing Settings → Capabilities 打开 SpatialPerception 选项，如下图所示：



运行后，我们观察周围环境，会发现程序在扫描周围环境的过程中，会在环境表面生成网格，如下图所示：



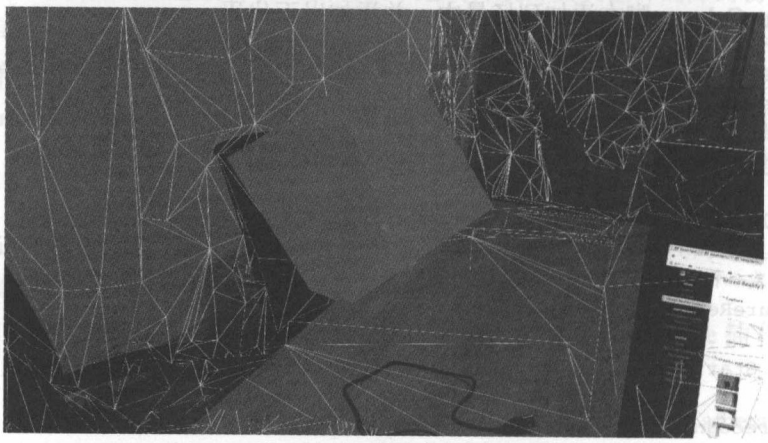
接下来，我们结合手势捕捉的功能实现；当用户双击的时候，创建一个立方体，并为立方体添加刚体，让立方体受重力作用。

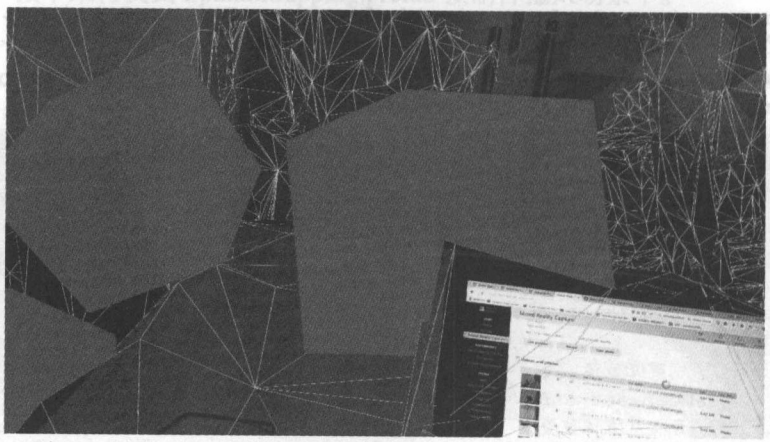
新建 CreateCube.cs 脚本添加到场景中，并添加以下代码：

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.VR.WSA.Input;

public class CreateCube : MonoBehaviour
{
    //定义一个手势识别器
    GestureRecognizer recognizer;

    void Start ()
    {
        recognizer = new GestureRecognizer();
        //设置识别类型，这里为识别单击和双击
        recognizer.SetRecognizableGestures(GestureSettings.Tap |
        GestureSettings.DoubleTap);
        recognizer.TappedEvent += TapEventHandler;
        recognizer.StartCapturingGestures();
    }

    //手势识别事件
    void TapEventHandler(InteractionSourceKind source, int tapCount, Ray headRay)
    {
        //双击生成 Cube
        if (source == InteractionSourceKind.Hand && tapCount == 2)
        {
            //创建立方体
            GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);
            //设置立方体位置
            cube.transform.position = new Vector3(0, 0, 2);
            //设置立方体大小
            cube.transform.localScale = new Vector3(0.3f, 0.3f, 0.3f);
            //为立方体添加刚体
```

```
            Rigidbody body = cube.AddComponent<Rigidbody>();

        }
    }

    void OnDestroy()
    {
        //停止捕捉手势
        recognizer.StopCapturingGestures();
        //取消手势事件
        recognizer.TappedEvent -= TapEventHandler;
    }

}
```

导出运行之后，双击创建立方体，我们可以从下图中看到，立方体在掉落的过程中，与真实的椅子和桌面发生碰撞并停止掉落，这就是空间映射的功能。





《计算机视觉增强现实应用概论》

专家顾问

池哲儒 谈飞 胡进臻 黄均昕 吕树炎

《计算机视觉增强现实应用平台开发》

专家顾问

杨发文 张天夫 郭翔宇 李森林

《计算机视觉增强现实美术内容设计》

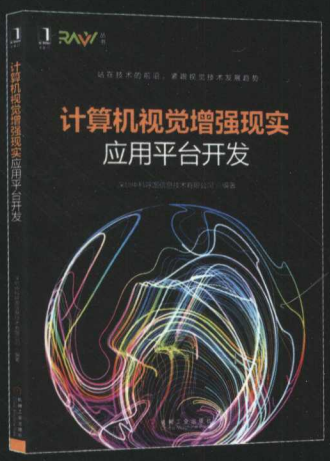
专家顾问

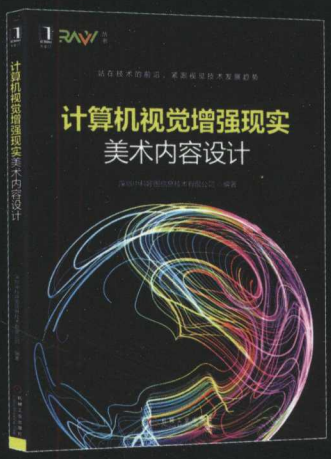
蒋斌 付旭耀 胡小亮

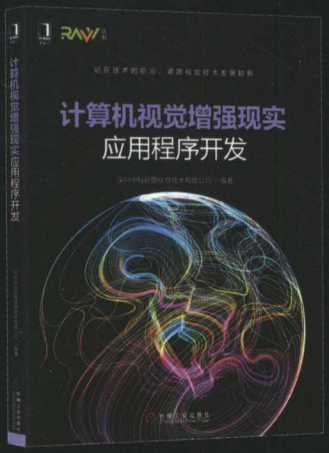
《计算机视觉增强现实应用程序开发》

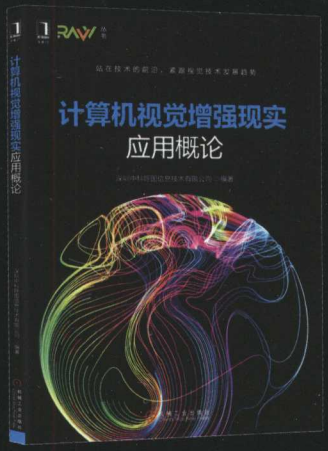
专家顾问

蒋斌 胡小亮 付旭耀









上架指导：计算机/程序设计

ISBN 978-7-111-57690-7

定价：49.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn